VŠB – Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Rozpoznávání objektů a jejich polohy na základě 3D modelu

Object Recognition Based on the 3D Models

2020 Michal Břemek

VŠB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Zadání diplomové práce

Student: **Bc. Michal Břemek**

Studijní program: N2647 Informační a komunikační technologie

Studijní obor: 2612T025 Informatika a výpočetní technika

Téma: Rozpoznávání objektů a jejich polohy na základě 3D modelu
Object Recognition Based on the 3D Models

Jazyk vypracování: čeština

Zásady pro vypracování:

Rozpoznávání objektů na základě 3D modelů může hrát významnou roli v úlohách rozšířené reality, kdy často, s ohledem na předpokládaný větší počet typů objektů, nelze využít postupů založených na klasickém učení využívajícím většího množství možných pohledů na 3D objekt. V diplomové práci proveďte následující:

1. Seznamte se přiměřeně s kontextem vaší budoucí práce (rozpoznávání na základě 3D modelu). Seznamte se s knihovnou [1].
2. S využitím uvedené knihovny realizujte software pro rozpoznávání objektů z 3D map. Vyhodnoťte získané zkušenosti.
3. Na základě zkušeností dle předchozího bodu uvažte možnost modifikace vybraných metod z knihovny s cílem dosáhnout urychlení rozpoznání (nejlépe v reálném čase) nebo s cílem zvětšení přesnosti rozpoznání.
4. Realizovaný software důkladně experimentálně ověřte. Popište nedostatky (a také přednosti) zvoleného postupu. Zvažte, zda a jak by případné nedostatky bylo možné odstranit.
5. Vše pečlivě zdokumentujte v textové části práce. Implementaci proveďte v C/C++.

Seznam doporučené odborné literatury:

[1] Point Cloud Library, http://pointclouds.org/. Dále články odkazované z popisu jednotlivých metod uvedené knihovny.

Formální náležitosti a rozsah diplomové práce stanoví pokyny pro vypracování zveřejněné na webových stránkách fakulty.

Vedoucí diplomové práce: **doc. Dr. Ing. Eduard Sojka**

Datum zadání: 01.09.2019

Datum odevzdání: 15.05.2020

doc. Ing. Jan Platoš, Ph.D.
vedoucí katedry

prof. Ing. Pavel Brandštetter, CSc.
děkan fakulty

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

V Hlučíně, dne 14. 5. 2020 .....

Rád bych na tomto místě poděkoval své rodině za podporu a doc. Dr. Ing. Eduardovi Sojkovi za odborné a vstřícné vedení diplomové práce.

Abstrakt

Diplomová práce se zabývá rozpoznáváním většího počtu objektů a odhadem jejich pózy na základě jejich 3D modelu. Rozpoznávání probíhá z 3D map získaných z různých hloubkových senzorů. Cílem práce je realizovat software pro rozpoznávání pomocí nejmodernějších metod z knihovny Point Cloud Library a pokusit se vylepšit některý krok z rozpoznávacího řetězce. S využitím neuronové sítě jsem se rozhodl modifikovat metodu pro hledání korespondujících párů klíčových bodů. V práci popisuji motivaci použití právě neuronové sítě, její výhody i nevýhody. Na závěr jsou experimentálně otestovány některé její parametry. Nechybí ani porovnání neuronové sítě s klasickou metodou.

Klíčová slova: PCL, mračno bodů, 3D model, Dlib, neuronová síť, detekce objektů, odhad pózy, klíčové body, FPFH, C/C++

Abstract

The thesis deals with the recognition of a larger number of objects and the estimation of their poses based on their 3D model. Recognition takes place from 3D maps obtained from various depth sensors. The goal of this work is to implement software for recognition using state of the art methods included in the Point Cloud Library and try to improve any step of the recognition chain. Using a neural network, I decided to modify the method for finding corresponding pairs of key points. I describe the motivation for the use of a neural network, its advantages and disadvantages. Finally, some of its parameters are experimentally tested. There is also a comparison of the neural network with the classical method.

Key Words: PCL, point cloud, 3D model, Dlib, neural network, object detection, pose estimation, key points, FPFH, C/C++

Obsah

Seznam použitých zkratek

2D	–	2 dimenze
3D	–	3 dimenze
6DoF	–	Six degrees of freedom
CAD	–	Computer aided design
CPU	–	Central processing unit, Procesor
CSV	–	Comma separated values
CUDA	–	Compute unified device architecture
FPFH	–	Fast point feature histograms
FPS	–	Frames per second
GPU	–	Graphics processing unit, Grafický procesor
NaN	–	Not a number
PCA	–	Principal component analysis
PCD	–	Point cloud data
PCL	–	Point cloud library
PFH	–	Point feature histograms
PLY	–	Polygon file format
RGB	–	Red-Green-Blue
RGB-D	–	Red-Green-Blue-Depth
USB	–	Universal serial bus

Seznam obrázků

Seznam tabulek

Seznam zdrojových kódů

Úvod

Rozpoznávání objektů je technika počítačového vidění pro detekci a identifikaci objektů v obraze či videosekvenci. Lidé dokážou rozpoznat vícero objektů ve scéně bez větších potíží, a to bez ohledu na velikost obrazu či jeho deformaci. Problém nemáme ani s objekty, které jsou částečně skryty, to je oblast, ve které systémy počítačového vidění stále velice pokulhávají. S příchodem nové generace hloubkových senzorů a neustále rostoucím výkonem spolu s poklesem ceny se stává použití trojrozměrných dat stále oblíbenější. Čím dál tím více lidí tak má přístup k 3D datům v reálném čase. Přesná, rychlá, robustní, škálovatelná a snadno trénovatelná metoda, která řeší tento úkol, bude mít velký dopad v aplikačních oblastech, jako je robotika nebo rozšířená realita.

Práce je rozdělena na pět hlavních kapitol. V kapitole 1 se podíváme na nástroje, které budeme v aplikaci potřebovat. To zahrnuje jak knihovny (PCL a Dlib), tak i programovací techniky, datové formáty, datové sady a použitý hloubkový senzor.

Kapitola 2 popisuje nutnou teorii k vytvoření úplné aplikace pro rozpoznávání většího počtu objektů ve scéně. Zároveň kapitola popisuje implementaci jednotlivých kroků rozpoznávacího řetězce pomocí knihovny PCL. Teorie je doplněná jak o obrázky, tak o ukázkové zdrojové kódy v jazyce C/C++.

Hlavním cílem práce je pokusit se jeden z těchto kroků vylepšit. Proto se v kapitole 3 podíváme na způsob výběru korespondenčních bodů pomocí neuronové sítě. Řekneme si, jak by tento způsob mohl vylepšit celkový výkon rozpoznávání oproti běžnějšímu výběru korespondencí pomocí euklidovské vzdálenosti (viz kapitola 2.7). Také se podíváme na výhody a nevýhody použití neuronové sítě a na způsob získávání trénovací množiny.

V kapitole 4 se blíže podíváme na návrh samotné aplikace a ukážeme si její použití na konkrétních datech. Podíváme se zde i na základní nastavení jednotlivých parametrů.

Kapitola 5 se věnuje experimentálnímu ověření realizované aplikace. Otestujeme, jestli neuronová síť oproti běžným metodám opravdu přinesla zlepšení. Vyhodnotíme výkon aplikace při různém nastavení některých parametrů. Zjistíme, jaký vliv na rozpoznávání mají různé trénovací množiny a popíšeme si případné nedostatky a přednosti zvoleného postupu.

1 Použité nástroje

1.1 Point Cloud Library

Point Cloud Library (1) je rozsáhlá open-source knihovna pro zpracování 2D/3D obrazu a mračen bodů. Obsahuje řadu nejmodernějších (*State of The Art*) algoritmů jako například filtrování, výpočet příznaků, rekonstrukci povrchu, registraci a segmentaci. Knihovna obsahuje více-vláknovou implementaci některých svých metod a algoritmů. Využívá k tomu open-source aplikační programovací prostředí OpenMP, což je projekt zabývající se multiprocesorovým programováním. Autoři knihovny uvádějí, že u osmijádrového procesoru mohou být výpočty 6-8x rychlejší. Knihovna je rozdělena na řadu menších knihoven, které mohou být kompilovány nezávisle. PCL je dostupná zdarma pro komerční i vědecké účely.

V zásadě už v sobě tato knihovna obsahuje veškeré metody potřebné pro funkční rozpoznávání a stačí je jen správně použít. Proto je to pro začátek ideální volba. Některé algoritmy knihovny je možné spustit na grafických kartách Nvidia s jádrem Fermi (CC 2.x) a vyšší a využít tak sílu paralelních výpočtů pomocí CUDA. Ke spuštění je ovšem potřeba dodatečná konfigurace, a nastavení je dost náročné, proto se ve své práci paralelizací na GPU zabývat nebudu.

1.2 Dlib

Dlib (2) je moderní volně dostupná knihovna s algoritmy a nástroji strojového učení sloužící pro vytváření komplexního softwaru v jazyce C++ pro řešení problémů reálného světa. Vývoj probíhá od roku 2002 a v současnosti již obsahuje softwarové komponenty pro práci s vlákny, datových struktur, lineární algebry, strojového učení, zpracování obrazu a další. Používá se jak v průmyslu, tak i v akademické sféře, a to v celé řadě oblastí včetně robotiky, mobilních telefonů a prostředí s velkým výpočetním výkonem.

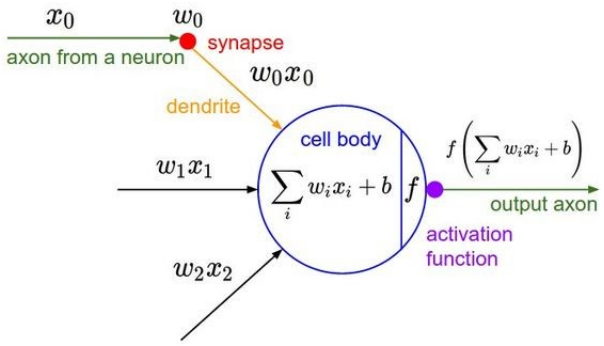
1.3 Neuronové sítě

Neuronové sítě (3) jsou inspirovány biologickými neuronovými sítěmi a snaží se napodobovat chování našeho mozku, jehož nejdůležitější vlastností je schopnost učení. To dělá z NS zatím nejvhodnějšího kandidáta pro vývoj umělé inteligence. Nicméně je zřejmé, že vytvoření umělého lidského mozku se všemi jeho schopnostmi je věc velice obtížná.

Neuronové sítě využívají distribuované, paralelní zpracování informace při provádění výpočtů. Ukládání, zpracování a předávání informace probíhá prostřednictvím celé neuronové sítě, spíše než pomocí určitých paměťových míst. Znalosti jsou ukládány především prostřednictvím síly vazeb mezi jednotlivými neurony. Vazby mezi neurony vedoucí ke „správné odpovědi“ jsou posilovány a naopak, vazby vedoucí ke „špatné odpovědi“ jsou oslabovány pomocí opakované expozice příkladů popisujících problémový prostor. Odpadá tedy nutnost algoritmizace úlohy, které je nahrazena předložením trénovací množiny neuronové síti a jejím učením.

1.3.1 Stavba neuronu

Základní stavební jednotkou neuronových sítí je neuron. Z něho se dále tvoří vrstvy a z nich je poskládaná celá síť. Uměle vytvořený neuron je inspirován svým biologickým vzorem a snaží se zjednodušeně napodobit jeho dendrity, tělo buňky, axonové vlákno a synapse. Model neuronu lze schematicky vyjádřit asi takto:



Obrázek 1: Schématický model neuronu.

Kde w_i jsou synaptické váhy upravující vstupní signál neuronů. x_i jsou výstupní signály předchozích neuronů. Hodnota b (tzv. *bias*) se přičítá k celkové sumě všech vstupů a posunuje tak aktivační funkci po ose x. f je aktivační funkce neuronu, jehož výstup je zároveň i výstupem neuronu.

1.3.2 Aktivační funkce

Aktivační funkce je důležitou součástí neuronu, která určí, zda byl neuron excitován neboli aktivován. Excitovaný neuron posílá svůj výstup neuronům, které jsou na něj napojeny. Uvedeme si některé základní a často používané funkce. Grafy těchto funkcí jsou znázorněny, viz Obrázek 2.

Funkce **Signum** neboli ostrá nelinearita je skoková funkce, pro kterou platí: Je-li vstupní hodnota větší než určený práh, bude funkce nabývat hodnoty 1, v opačném případě 0. Funkce se hodí pouze pro binární klasifikace (Perceptron).

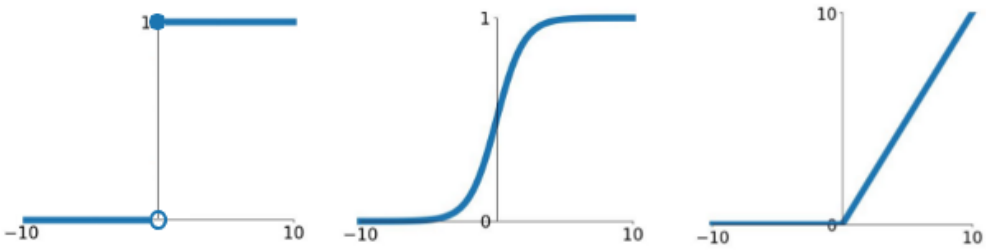
$$f(x) = \begin{cases} 1 & pokud \quad x \geq 0 \\ 0 & pokud \quad x < 0 \end{cases} \tag{1}$$

Funkce **Sigmoid** je spojitá funkce, jejíž hodnoty leží v intervalu od 0 do 1. Je tedy vhodná pro klasifikaci více než dvou tříd. Funkce je vyjádřena níže. (λ je strmost sigmoidu)

$$f(x) = \frac{1}{1+e^{-\lambda x}} \tag{2}$$

Funkce **ReLu** (*Rectified Linear Units*) je velmi využívána v konvolučních neuronových sítích a je jednoduší pro výpočet než funkce Sigmoid. Funkci lze zapsat následovně:

$$f(x) = \begin{cases} x & pokud \quad x \geq 0 \\ 0 & pokud \quad x < 0 \end{cases} \tag{3}$$



Obrázek 2: Aktivační funkce zleva: Signum, Sigmoid a ReLu.

1.3.3 Dopředné šíření

Dopředné šíření signálu (*Feedforward*) je základní schopností neuronové sítě. Tímto způsobem získáme odezvu NS na vstupní data.

Průběh dopředného šíření:

1. Nejprve jsou excitovány na odpovídající úroveň neurony vstupní vrstvy.
2. Tyto excitace jsou pomocí vazeb přivedeny k následující vrstvě a upraveny (zesíleny či zeslabeny) pomocí synaptických vah.
3. Každý neuron této vyšší vrstvy provede sumaci upravených signálů od neuronů nižší vrstvy a je excitován na úroveň danou svou aktivační funkcí.
4. Tento proces probíhá přes všechny vnitřní vrstvy až k vrstvě výstupní, kde pak získáme excitační stavy všech jejích neuronů.

1.3.4 Backpropagation

Metoda, která umožňuje adaptaci neuronové sítě nad danou trénovací množinou, se nazývá backpropagation (v překladu: metoda zpětného šíření). Metoda spočívá v šíření informace směrem od vrstev vyšších k vrstvám nižším na rozdíl od dopředného šíření.

Průběh backpropagation:

1. Vezmeme nejprve vektor I_i i-tého prvku trénovací množiny, kterým excitujeme neurony vstupní vrstvy na odpovídající úroveň.
2. Známým způsobem provedeme dopředné šíření tohoto signálu až k výstupní vrstvě neuronů.
3. Srovnáme požadovaný stav daný vektorem O_i i-tého prvku trénovací množiny se skutečnou odezvou neuronové sítě.
4. Rozdíl mezi skutečnou a požadovanou odezvou definuje chybu neuronové sítě. Tuto chybu pak v určitém poměru (*Learning Rate*) „vracíme zpět“ do neuronové sítě formou úpravy synaptických vah mezi jednotlivými vrstvami směrem od horních vrstev k vrstvám nižším tak, aby chyba při následující odezvě byla menší.
5. Po vyčerpání celé trénovací množiny se vyhodnotí celková chyba přes všechny vzory trénovací množiny a pokud je tato chyba vyšší než požadovaná, celý proces se opakuje znovu.

Podstata metody backpropagation spočívá v hledání minima funkce chyby E definované např. tímto způsobem:

$$E = \frac{1}{2}\sum_{i=1}^{p}\sum_{j=1}^{m}\left(y_j - o_j\right)_i^2 \quad (4)$$

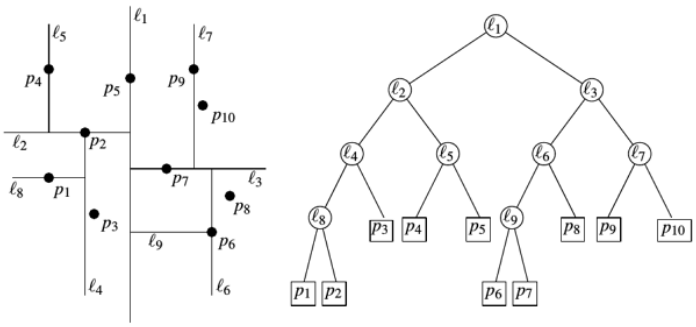
Kde y_j je skutečná odezva j-tého neuronu výstupní vrstvy. o_j je požadovaná odezva j-tého neuronu výstupní vrstvy daná vzorem trénovací množiny. p je celkový počet vzorů trénovací množiny. m je počet neuronů výstupní vrstvy.

1.4 K-d strom

K-d strom (4) (*k-d tree*) je datová struktura, která se používá k organizaci množiny bodů v k-rozměrném prostoru. Jedná se v podstatě o multidimenzionální binární strom, velice efektivní pro operace vyhledávání rozsahu a nalezení nejbližšího souseda. K-d strom rekurzivně dělí prostor tak, že jednotlivé osy dělí nadrovinou kolmou k dané ose. Bod, ve kterém je osa rozdělena, je mediánem souřadnic bodů daného podintervalu. Tímto postupem se postupně dopracujeme až k situaci, kdy je strom vyvážený a rozděluje prostor na podčásti, které obsahují pouze jeden bod anebo podstatně redukovanou množinu bodů.

Každý uzel v k-d stromu je k-dimenzionální bod. Každý uzel, který není listem, může být považován za rozdělující rovinu, která dělí prostor na dvě poloviny a je kolmá na osu dimenze, kterou rozděluje. Body, které jsou v levé části, jsou ve stromu reprezentovány větví ležící nalevo, body napravo pak tvoří pravou část podstromu.

K-d strom s n body má prostorovou složitost $O(n)$ a může být sestaven v čase $O(n \log n)$. Časová složitost při hledání nejbližšího souseda pro 2D je v nejhorším případě $O\left(\sqrt{n}\right)$, průměrná časová složitost je $\Theta(\log n)$.



Obrázek 3: Ukázka rozdělení prostoru k-d stromem a samotný k-d strom.

1.5 Datový formát PLY

Datový formát pro polygony (*Polygon File Format*) vyvinuli na Stanfordské univerzitě. Byl navržen především pro ukládání prostorových dat z 3D skenerů. Formát ukládání podporuje relativně jednoduchý popis objektu jako list polygonů. Do souboru lze také uložit různé vlastnosti včetně barvy, průhlednosti, povrchových normál a souřadnic textur. Soubor lze uložit buď čitelně pro člověka v ASCII (viz Zdrojový kód 1) nebo binárně.

```
ply
format ascii 1.0
comment this file is a three-sided pyramid
element vertex 4
property float x
property float y
property float z
element face 4
property list uchar int vertex_index
end_header
0 0 0           { start of vertex list }
0 0 1
0 1 0
1 0 0
3 0 1 2         { start of face list }
3 0 1 3
3 0 2 3
3 1 3 2
```

Zdrojový kód 1: Ukázka 3D elementu uloženého v PLY formátu pomocí ASCII.

1.6 Datový formát PCD

Datový formát PCD (*Point Cloud Data*) byl vytvořen autory knihovny PCL za účelem doplnění stávajících formátů (PLY, STL, OBJ, …). Ty mají několik nedostatků, protože byli vytvořeny pro jiné účely a v době, kdy ještě nebyly snímací zařízení na takové technické úrovni, na jaké jsou dnes. Oproti starším formátům dokáže uchovávat a zpracovávat datové sady organizovaných mračen bodů. To je velmi důležité například pro aplikace v reálném čase. Dovoluje ukládání různých datových typů, přičemž podporuje všechny primitivní typy (char, short, int, float, double). Neplatné dimenze jsou uloženy jako NAN. Formát také nabízí možnost ukládání N-dimenzionálních histogramů pro deskriptory.

```
# .PCD v.7 - Point Cloud Data file format
VERSION .7
FIELDS x y z rgb
SIZE 4 4 4 4
TYPE F F F F
COUNT 1 1 1 1
WIDTH 213
HEIGHT 1
VIEWPOINT 0 0 0 1 0 0 0
POINTS 213
DATA ascii
0.93773 0.33763 0 4.2108e+06
0.90805 0.35641 0 4.2108e+06
0.81915 0.32 0 4.2108e+06
```

Zdrojový kód 2: Ukázka mračna bodů uloženého v PCD formátu pomocí ASCII.

1.7 BOP: Benchmark for 6D Object Pose Estimation

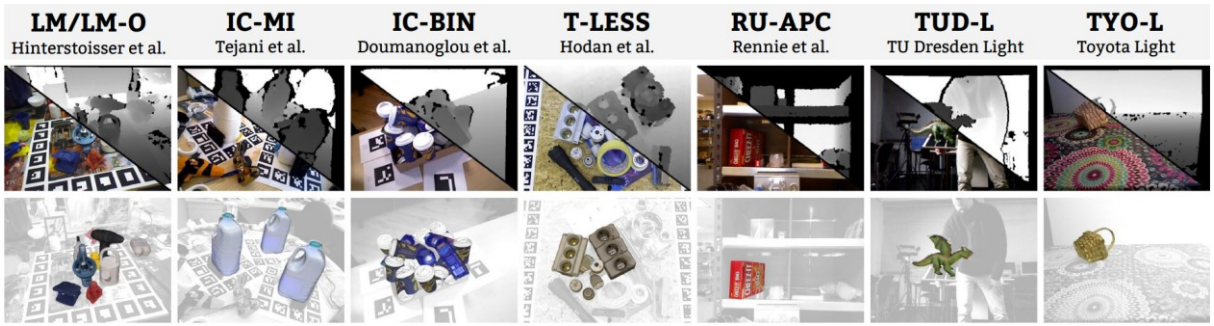
Cílem BOP (5) je zachytit dosaženou úroveň vývoje při odhadování 6D pozice, tj. 3D posuvu a 3D rotace, pevných objektů (*Rigid Objects*) v RGB / RGB-D obrazech. Benchmark tak přináší možnost porovnání kvality jednotlivých metod.

Benchmark zahrnuje:

- Datasety uložené v jednotném formátu, které pokrývají různé praktické scénáře.
- Metodiku hodnocení správnosti určených póz.
- Online systém hodnocení, který je otevřený pro předkládání nových metod.

1.7.1 Datasety

Benchmark v současné době obsahuje 11 různých datasetů pokrývající odlišné aplikační scénáře. Každý dataset obsahuje 3D modely objektů, tréninkové a testovací RGB-D obrazy s anotovanými pozicemi objektů (*Ground Truth*) a vlastními parametry kamery. 3D objekty modelů byly vytvořeny pomocí systému typu CAD nebo byly naskenovány a prošly 3D rekonstrukcí povrchu. Trénovací obrázky ukazují jednotlivé objekty z různých pohledů a jsou buď nasnímány snímačem RGB-D, Gray-D nebo jsou synteticky vytvořené. Testovací snímky jsou zachyceny ve scénách s různou mírou složitosti, šumu nebo okluze. Níže je vyobrazeno 8 datasetů (viz Obrázek 4), které se používaly v benchmarku v roce 2018. Později přibyly ještě ITODD, Homebrewed a YCB-Video.



Obrázek 4: Datasety používáné v BOP 2018.

Veškeré datasety jsou dostupné na webové stránce https://bop.felk.cvut.cz/datasets/.

1.7.2 Metodika hodnocení

Chyba odhadované pózy $\hat{P}$ vůči póze anotované $\bar{P}$ pro daný model O je měřena pomocí tří funkcí pro stanovení chyby pozice. Tyto funkce jsou definovány níže. Jejich implementace je k dispozici v BOP Toolkit na https://github.com/thodan/bop_toolkit.

Visible Surface Discrepancy (VSD) (5):

$$e_{VSD}(\hat{S},\bar{S},S_I,\hat{V},\bar{V},\tau) = avg_{p\in\hat{V}\cup\bar{V}}\begin{cases}0 & if\ p\in\hat{V}\cap\bar{V}\wedge\left|\hat{S}(p)-\bar{S}(p)\right|<\tau\\ 1 & otherwise\end{cases} \quad (5)$$

$\hat{S}$ a $\bar{S}$ jsou mapy vzdáleností získané vykreslením modelu O v odhadované póze $\hat{P}$ a anotované póze $\bar{P}$. Tyto mapy vzdáleností jsou porovnány s mapami vzdáleností S_I testovaného obrázku I pro získání masek viditelnosti $\hat{V}$ a $\bar{V}$ (tj. binární obrazy, kde je model O viditelný na obrázku I). τ je tolerance výchylky.

Maximum Symmetry-Aware Surface Distance (MSSD) (6):

$$e_{MSSD} = min_{S \in S_O} max_{x \in O} \left\| \hat{P}x - \bar{P}Sx \right\|_2 \tag{6}$$

S_O je sada symetrických transformací modelu O.

Maximum Symmetry-Aware Projection Distance (MSPD):

$$e_{MSPD} = min_{S \in S_O} max_{x \in O} \left\| proj(\hat{P}x) - proj(\bar{P}Sx) \right\|_2 \tag{7}$$

Kde $proj$ je operace 2D projekce (výsledek je v pixelech).

Výkon metody M pro každou chybovou funkci X je měřen jako průměrné vyvolání (*Average Recall*) AR_X, definované jako zlomek anotovaných instancí objektů, pro které byla odhadnuta správná pozice. Odhad pózy je považován za správný, pokud platí $e_X < \theta$. Práh správnosti θ nabývá hodnot od 5% do 50% s krokem 5%. Celkový výkon metody je měřen následovně:

$$AR = (AR_{VSD} + AR_{MSSD} + AR_{MSPD})/3 \tag{8}$$

Do měření se započítávají pouze instance objektů, které jsou viditelné alespoň z 10%.

1.8 Hloubková kamera RealSense D435

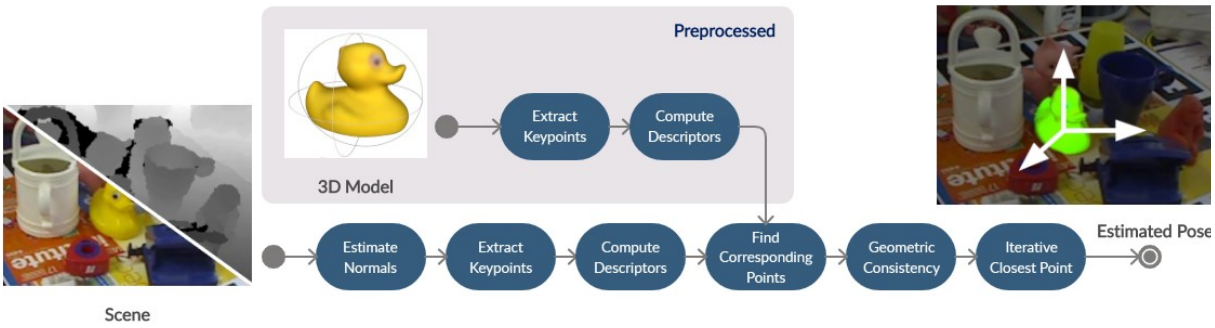
RealSense (7) je technologie vyvíjena firmou Intel. Kamery typu D435 mají zajištěnou konektivitu i napájení pomocí jediného kabelu USB 3.0. Pro generování 3D obrazu využívají technologii stereo vidění, tedy dvě kamery s rozlišením 1 200 × 720 při 90 FPS, které sledují aktivně vysílané infračervené záření. Kamera je schopna snímat hloubku obrazu v rozsahu 0.105 – 10 m. Přesnost této hodnoty je závislá na kalibraci a světelných podmínkách. Pro snímání barevného spektra je použita RGB kamera s rozlišením 1 920 × 1 080 pixelů při 30 FPS.

Obrázek 5: Hloubková kamera RealSense D435.

V aplikaci nicméně nebudu streamovat data přímo z kamery. Místo toho použiji dostupný software RealSense Viewer (8) a pomocí něho si nasnímám a uložím hloubková data do PLY formátu.

2 Průběh rozpoznávání

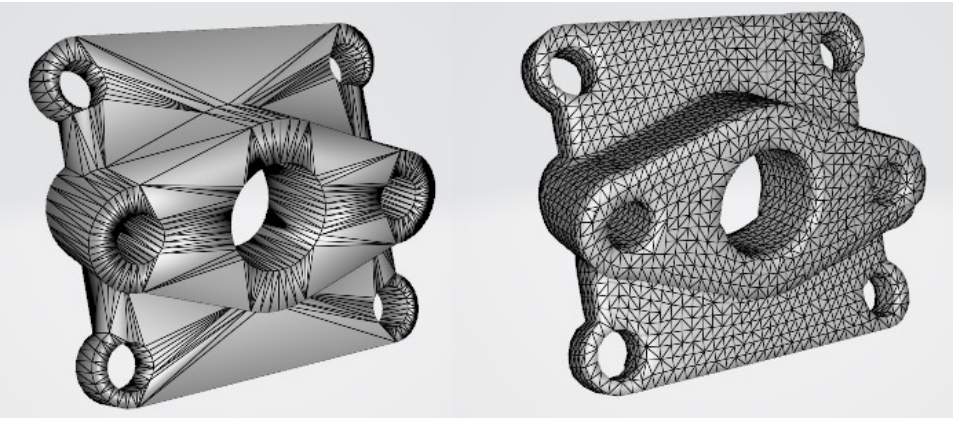
V následujících kapitolách se podíváme na postup realizace systému rozpoznávání s využitím lokálních deskriptorů. Vstupem aplikace budou mračna bodů modelů (tj. 3D modely) a mračno bodů prohledávané scény. Výstupem by měla být identifikace těchto objektů společně s jejich pozicí ve scéně. Můžeme toho dosáhnout pomocí následujícího algoritmu (viz Obrázek 6): Nejprve provedeme extrakci klíčových bodů (kapitola 2.5) a vypočítáme jejich deskriptory (kapitola 2.6). Poté pro každý hledaný objekt provedeme následující: S použitím deskriptorů získáme sadu všech korespondujících bodů mezi modelem a scénou (kapitola 2.7). Pomocí shlukování korespondencí (kapitola 2.8) najdeme největší podmnožinu korespondencí, která je konzistentní (můžeme najít i více podmnožin). Někdy je ještě nutné výsledek dále zpřesnit a odfiltrovat i případné chybné nálezy (kapitola 2.9).



Obrázek 6: Průběh rozpoznávání.

2.1 Předzpracování objektů

Modely objektů, které budeme používat pro hledání můžou pocházet z různých 3D modelovacích nástrojů (Blender, AutoCAD, Inventor, …), popřípadě to můžou být i skeny reálných předmětů. Pro naše potřeby musíme některé modely upravit, aby nám více vyhovovaly. Zejména potřebujeme, aby měly uniformě rozložené vertexy, správně orientované normály a pokud možno obsahovaly pouze vrstvu vertexů viditelnou z vnější strany. Výsledné modely budeme potřebovat v PLY formátu i s normálami. K takovým topologickým úpravám nám dopomůžou aplikace třetích stran MeshLab a Instant Meshes. Ukázka, jak může vypadat upravený a neupravený model, viz Obrázek 7.



Obrázek 7: Model objektu z datasetu ITODD. Vlevo – původní CAD model. Vpravo – remesh modelu.

MeshLab – Open source program se zaměřením na zpracování a editaci 3D trojúhelníkových sítí. Poskytuje metody pro jejich editaci, zjednodušení, texturování, konverzi mezi formáty a jiné.

Instant Meshes – Také open source program. Slouží k překreslení topologie vertexů (*Remesh*).

Dalším krokem bude spočítání klíčových bodů a jejich deskriptorů (popsáno v kapitole 2.5 a 2.6) Na rozdíl od scény, která se každým snímkem aktualizuje, hledané modely jsou neměnné. Je proto logické, že modely budeme zpracovávat pouze jednou po spuštění programu. V případě, že dopředu víme, jaké nastavení parametrů budeme používat, můžeme si modely předzpracovat zvlášť a při spuštění je jen patřičně načíst.

2.2 Načtení mračna bodů

Způsob, jakým načteme mračno bodů do příslušné datové struktury pcl::PointCloud<T>, se odvíjí od formátu daného souboru. Pokud se jedná o formát, který podporuje ukládání 3D dat, tj. formáty PCD, PLY nebo jiné, tak nám bude stačit použít příslušný vstupně/výstupní modul z knihovny PCL. Jak modely, tak i scény mohou být uloženy v těchto formátech. Konkrétní datové body mohou nabývat podle potřeby těchto typů: pcl::PointXYZ, pcl::PointXYZRGB, pcl::PointXYZRGBA a další.

Druhá varianta ukládání je pomocí hloubkových map, kde máme na vstupu obrázek s informacemi o vzdálenostech pixelů od senzoru a parametry použité kamery (*Camera Intrinsic*). Pomocí parametrů z kamery dokážeme převést 2D hloubkovou mapu na požadované 3D mračno bodů. Tento způsob ukládání scén je použit u datasetů z BOP. Některé scény mohou být oříznuté a nemusí tak být vycentrované správně, a proto je potřeba spočítat potřebnou korekci *yOff*. Převod je znázorněn v následujícím zdrojovém kódu:

```
float yOff = (2 * camera.cy - image.height) / camera.fy;
for (int y = 0; y < image.height; y++){
   for (int x = 0; x < image.width]; x++){
       float depth = image.at(y, x) * depthScale;
       point->x = (x - camera.cx) * depth / camera.fx;
       point->y = (y - camera.cy) * depth / camera.fy + (yOff * depth);
       point->z = depth;
   }
}
```

Zdrojový kód 3: Převod 2D hloubkové mapy na mračno bodů.

2.3 Filtrace a podvzorkování

Mračno bodů, které se nám povedlo načíst, budeme muset nejprve trochu upravit. Určitě se potřebujeme zbavit tzv. NaN bodů, které mohly vzniknout například tak, že senzor vyhodnotil daný bod, jako bod v nekonečnu. K tomu nám v PCL slouží pcl::removeNaNFromPointCloud.

Dále by nám mohly dělat problémy i některé osamocené body. Například bychom u nich nedokázali vypočítat normály. K tomu použijeme metodu pcl::RadiusOutlierRemoval (viz Zdrojový kód 4). Musíme zde určit, jaký chceme minimální počet sousedících bodů *MIN_NEIGHBORS* ve zvoleném poloměru *RADIUS*.

```
pcl::RadiusOutlierRemoval<PointType> outrem;
outrem.setInputCloud(cloud);
outrem.setRadiusSearch(RADIUS);
outrem.setMinNeighborsInRadius(MIN_NEIGHBORS);
outrem.filter(*outCloud);
```

Zdrojový kód 4: Odstranění odlehlých bodů v knihovně PCL.

Abychom snížili požadavky na výpočetní výkon, můžeme provést podvzorkování (*Downsampling*). To je proces, při kterém dojde ke snížení počtů bodů v mračně. V aplikaci používám voxelizaci. Při ní se vytvoří 3D mřížka, jehož nejmenší 3D boxy se nazývají voxely. Jejich velikost je volitelná. Můžeme říct, že se jedná o velikost listu. Pro každý voxel zprůměrujeme všechny jeho vnitřní body, a tím nám vznikne jediný bod tzv. centroid. Podvzorkování nám v pozdějších krocích ušetří nároky na výpočetní výkon při zachování dostatečné přesnosti. Ukázka použití:

```
pcl::VoxelGrid<PointType> vox;
vox.setInputCloud(cloud);
vox.setLeafSize(LEAF_SIZE, LEAF_SIZE, LEAF_SIZE);
vox.filter(*outCloud);
```

Zdrojový kód 5: Voxelizace mračna v knihovně PCL.

2.4 Normály

Normála je v geometrii přímka, která je kolmá na daný podprostor. Vektor určující směr normály nazýváme normálový vektor. V našem prostorovém případě hledáme vektory kolmé na rovinu. Správné určení normál je velice důležité, protože díky nim počítáme jak klíčové body, tak i jejich deskriptory.

U geometrického povrchu je obvykle triviální odvodit směr normály pro určitý bod na povrchu jako vektor kolmý k povrchu v tomto bodě. Ovšem v případě nespojitého mračna bodů máme k dispozice dvě možnosti. Buď získáme podkladový povrch pomocí technik rekonstrukce povrchu (*Mesh Reconstruction*), a poté určíme normály z tohoto povrchu. Anebo použijeme aproximace k odvození povrchových normál přímo z mračna bodů. Obecně je efektivnější použít tuto druhou variantu, protože rekonstrukce povrchu je časově náročný proces. Ačkoliv existují různé metody na odvození normál, jako nejjednodušší se jeví problém stanovení normál k bodu na povrchu aproximovat na problém odhadu normál tečné roviny k povrchu, čímž ve skutečnosti získáme problém odhadu nejmenších čtverců. Řešení nalezení povrchových normál je tímto zredukováno na analýzu vlastních vektorů a vlastních hodnot (*PCA – Principal Component Analysis*) kovarianční matice získané z nejbližších sousedů dotazovaného bodu. Konkrétně pro každý bod p_i sestavíme kovarianční matici C takto:

$$C = \frac{1}{k}\sum_{i=1}^{k}(p_i - \bar{p}) \cdot (p_i - \bar{p})^T \tag{9}$$

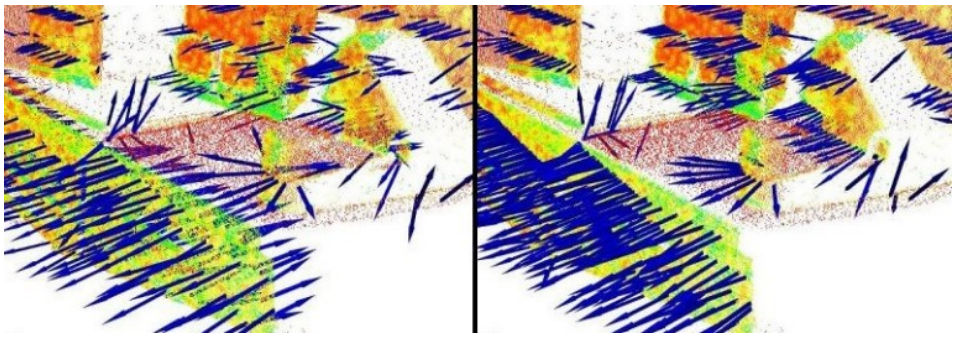
$$C \cdot \vec{v_j} = \lambda_j \cdot \vec{v_j}, \;\; j \in \{0,1,2\} \tag{10}$$

Kde k je počet sousedů z okolí bodu p_i. $\bar{p}$ reprezentuje 3D centroid nejbližších sousedů. λ_j je j-té vlastní číslo z kovarianční matice. $\vec{v_j}$ je j-tý vlastní vektor.

Jelikož neexistuje žádný matematický způsob, jak vyřešit znaménko normály, její orientace získaná pomocí PCA není jednoznačná ani konzistentní na celém mračně bodů. Řešení tohoto problému je triviální,

pokud tedy známe zorný bod snímače v_p. Abychom orientovali všechny normály $\overrightarrow{n_i}$ konzistentně směrem k tomuto zornému bodu, musíme splnit podmínku:

$$\overrightarrow{n_i} \cdot (v_p - p_i) > 0 \tag{11}$$



Obrázek 8: Vlevo – nekonzistentně orientované normály. Vpravo – výsledek po přeorientování normál vůči senzoru.

Velice důležité je určení správného okolí bodů. Snažíme se, aby co nejlépe odpovídalo scéně, ve které normály odhadujeme. Pokud zvolíme okolí příliš velké, zaniknou detaily. Příliš malé okolí může zase způsobit, že budou normály zkreslené vlivem šumu. Okolí bodu volíme buď pomocí poloměru, nebo pomocí počtu nejbližších sousedů. Zdrojový kód níže ukazuje, jak odhadnout normály v knihovně PCL:

```
pcl::NormalEstimation<PointType, NormalType> norm_est;
norm_est.setRadiusSearch(RADIUS);
norm_est.setInputCloud(cloud);
norm_est.compute(*outNormals);
```

Zdrojový kód 6: Odhad normál v knihovně PCL.

2.5 Klíčové body

Vypočítat deskriptory pro všechny body v mračnu by bylo výpočetně velmi náročné, a to dokonce i po podvzorkování. Proto se ve vstupních datech hledají takzvané klíčové body (*Key Points*), které jsou z hlediska zpracování obrazových dat nějak zajímavé. Základními vlastnostmi klíčových bodů je to, že takové body jsou stabilní, výrazné a zjistitelné z různých úhlů. Typicky je těchto bodů mnohonásobně méně, než jaký je celkový počet bodů ve scéně. Mezi algoritmy, jak získat tyto klíčové body v 3D prostoru patří například ISS, NARF, SIFT, SURF nebo Harris 3D.

Pro detekci klíčových bodů jsem se rozhodl pro Harrisův detektor. Myslím si, že tento detektor velmi spolehlivě označuje místa, které bych i já osobně označil jako klíčové viz Obrázek 9. Další výhodou je podle mě, velice intuitivní škálovatelnost detektoru pomocí dostupných parametrů. Více popsáno v následující kapitole.

Obrázek 9: Armadillo model – vybrané klíčové body pomocí Harrisova detektoru.

2.5.1 Harris 3D

Harrisův detektor (9) rohů se běžně používá v algoritmech strojového vidění k detekci rohů v obraze. Potřebujeme-li rozhodnout, zda bod v obraze představuje roh, zaměříme se na jeho okolí a sledujeme rozdíl intenzity barev ve všech směrech. Následující výpočty jsou čerpány ze zdroje (10). V případě tohoto detektoru konstruujeme v každém bodě obrazu matici:

$$M(x,y) = \begin{bmatrix} \langle (I_x)^2 \rangle & \langle I_x I_y \rangle \\ \langle I_x I_y \rangle & \langle (I_y)^2 \rangle \end{bmatrix} \tag{12}$$

$$\langle \psi(x,y) \rangle = \psi(x,y) * [G(x)G(y)] \tag{13}$$

$$G(u) = \frac{1}{\sigma\sqrt{2\pi}} exp\left(-\frac{u^2}{2\sigma^2}\right) \tag{14}$$

Volba hodnoty σ z (viz rovnice (14) ovlivňuje citlivost detektoru. K rozhodnutí, zda daný bod (x,y) lze považovat za roh, lze využít následující rovnice:

$$cor(x,y) = \det\big(M(x,y)\big) - 0{,}04 * trace^2\big(M(x,y)\big) \tag{15}$$

Bod (x, y) považujeme za roh, jestliže jsou splněny následující dvě podmínky:

- Hodnota cor(x,y) je větší než nějaká předem stanovená hodnota.
- Hodnota cor(x,y) je největší v okně o předem definovaných rozměrech.

Pro třetí dimenzi platí následující: Osa odpovídající nejmenší vlastní hodnotě se stane novou osou z:

$$z = f(x,y) = \frac{p_1}{2}x^2 + p_2 xy + \frac{p_3}{2}y^2 + p_4 x + p_5 y + p_6 \tag{16}$$

Nyní už se jedná o stejný dvourozměrný problém, jaký jsme popsali výše.

Následující kód ukazuje použití 3D Harrisova detektoru v PCL:

```
pcl::HarrisKeypoint3D<PointType, pcl::PointXYZI> detector;
detector.setInputCloud(cloud);
detector.setRadius(RADIUS);
detector.setThreshold(TRESHOLD);
detector.setMethod(pcl::HarrisKeypoint3D<PointType, pcl::PointXYZI>::HARRIS);
detector.setNormals(normals);
detector.setRefine(true);
detector.compute(*outKeypoints);
```

Zdrojový kód 7: Harris 3D v knihovně PCL.

Kde *RADIUS* ovlivňuje, z jak velkého okolí se budou klíčové body počítat. *TRESHOLD* ovlivňuje, při jaké hodnotě intenzity se už jedná o významný bod. Čím menší nastavíme treshold, tím více významných bodů nám detektor vrátí. Výsledné klíčové body, které získáme touto metodou, bude ale ještě potřeba upravit. Důvodem je, že klíčové body *outKeypoints* nejsou podmnožinou vstupního mračna bodů *cloud* a může se tak stát, že některé body budou ležet mimo toto mračno. To můžeme vyřešit pomocí k-d stromu, kde budeme hledat nejbližší bod, který leží v původním mračnu. Další problém je takový, že některé body jsou duplicitní. Je potřeba se této duplicity zbavit.

2.6 Deskriptory

Obecně deskriptory (*Descriptors*) slouží k popisu objektů a jejich základních charakteristik, jako jsou velikost, tvar, barva, textura a další. Ideální deskriptory musí být invariantní vůči translaci a rotaci. Musí být invariantní při změně měřítka. A také musí být odolné vůči šumu, který vzniká při nepřesném měření. V našem případě hledáme takové deskriptory, které popisují geometrii okolí jednotlivých klíčových bodů. Hlavní myšlenkou je identifikovat a popsat klíčové body z různých mračen bodů tak, abychom je byli schopni v dalším kroku navzájem porovnat. Dalo by se říct, že takovými jednoduchými deskriptory jsou i normály, ty ale o našem bodu nenesou dostatek informací a nesplňují tak podmínky k tomu, aby se daly považovat za dobré deskriptory. Nicméně nám můžou posloužit jako základ pro jejich výpočet.

Pokud deskriptory, pro daný bod, počítáme z celého mračna, tak se jedná o globální deskriptory. Pokud je počítáme jen z jeho okolí, tak mluvíme o lokálních deskriptorech. Z toho důvodu je časová složitost výpočtu lokálních deskriptorů lepší než těch globálních. Tedy O(nk) oproti $O(n^2)$, kde n je celkový počet bodů a k je počet bodů v okolí počítaného bodu. Navíc metody založené na lokálních deskriptorech dosahují lepších výsledků v přeplněných scénách. Z tohoto důvodu jsem se rozhodl, že budu dále pracovat pouze s lokálními deskriptory. Následující Tabulka 1 obsahuje výpis několika lokálních deskriptorů se zaměřením na rozpoznávání 3D objektů, popřípadě 3D registraci.

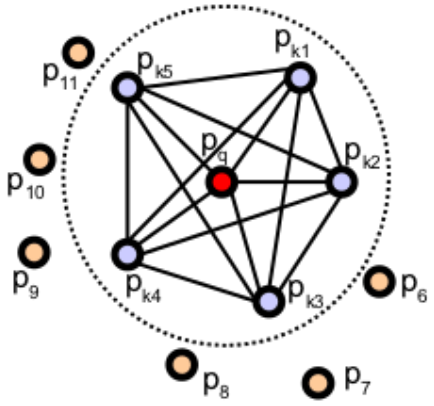
Tabulka 1: Výpis několika lokálních deskriptorů s jejich velikostmi a vlastnostmi. Zdroj: (11).

Deskriptor	Název	Velikost	Vlastnosti
SI	Spin Image (12)	153	Robustní proti okluzi a šumu.
3DSC	3D Shape Context (13)	1980	Překonává SI.
PFH	Point Feature Histogram (14)	125	Invariantní vůči pozici, rotaci a hustotě mračna bodů.
FPFH	Fast Point Feature Histogram (15)	33	Rychlejší varianta PFH.
NARF	Normal Aligned Radial Feature (16)	36	Invariantní vůči rotaci. Překonává SI.
SHOT	Signature of Histogram of Orientation (17)	352	Překonává SI.

Pro implementaci jsem si vybral deskriptor FPFH, protože se podle mého názoru pro tento problém hodí nejlépe. Jeden z důvodu je, že jeho velikost je oproti ostatním deskriptorům relativně malá, což může mít pozitivní vliv na rychlost v následujících krocích rozpoznávání. V následujících kapitolách se tedy více podíváme na deskriptory PFH a FPFH.

2.6.1 PFH

Cílem metody PFH (*Point Feature Histogram*) je reprezentovat geometrické vlastnosti okolí bodu (*k-neighborhood*) zobecněním průměrného zakřivení kolem tohoto bodu pomocí vícerozměrného histogramu hodnot. Tento vysoce dimenzionální hyperprostor nám poskytuje informativní popis pro reprezentaci rysů. Je invariantní k 6D pozici daného povrchu a velmi dobře se vyrovnává s různými hustotami vzorkování nebo hladinami šumu v okolí. Reprezentace PFH je založena na vztazích mezi body v sousedství a jejich povrchovými normálami. Jednoduše řečeno, snaží se co nejlépe zachytit změny navzorkovaného povrchu s přihlédnutím ke všem interakcím mezi směry odhadnutých normál. Výsledný hyperprostor je tedy závislý na kvalitě odhadu povrchových normál v každém bodě. Níže uvedený Obrázek 10 představuje diagram vlivové oblasti výpočtu PFH pro dotazový bod p_q, označený červenou barvou a umístěn uprostřed kruhu (koule ve 3D) s poloměrem r. Všechny jeho k sousední body (body se vzdáleností menší než poloměr r) jsou plně propojeny v síti. Konečný deskriptor PFH je počítán jako histogram vztahů mezi všemi páry bodů v k sousedství. Jeho výpočetní složitost je $O(k^2)$.



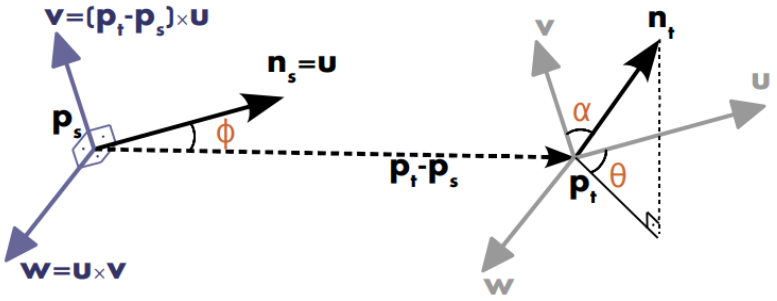
Obrázek 10: Diagram vztahů bodů z okolí bodu p_q, pro výpočet deskriptoru PFH.

Pro každý pár bodů p_i a p_j $(i \neq j, j < i)$ a jejich přidružené normály $\overrightarrow{n_i}$ a $\overrightarrow{n_j}$ zvolíme jeden bod jako p_s (*source*) a druhý jako p_t (*target*). Jako zdrojový bod p_s zvolíme ten bod, jehož normála svírá menší úhel se spojnicí obou bodů. V tomto bodě definujeme pevný souřadný rámec (*Darboux Frame*). Viz rovnice a obrázek:

$$\vec{u} = \overrightarrow{n_s} \tag{17}$$

$$\vec{v} = (p_t - p_s) \times \vec{u} \tag{18}$$

$$\vec{w} = \vec{u} \times \vec{v} \tag{19}$$



Obrázek 11: Znázornění Darbouxova rámce.

Pomocí vypočteného **uvw** rámce může být rozdíl mezi dvěma normálami $\overrightarrow{n_s}$ a $\overrightarrow{n_t}$ definován jako množina úhlových deskriptorů následovně:

$$d = \|p_t - p_s\|_2 \tag{20}$$

$$\alpha = \vec{v} \cdot \overrightarrow{n_t} \tag{21}$$

$$\varphi = \vec{u} \cdot \frac{(p_t - p_s)}{d} \tag{22}$$

$$\theta = \arctan(\vec{w} \cdot \overrightarrow{n_t}, \vec{u} \cdot \vec{v}) \tag{23}$$

Kde d je euklidovská vzdálenost mezi body p_t a p_s. Tímto způsobem získáme čtveřici $\langle \alpha, \varphi, \theta, d \rangle$ pro každý pár bodů z počítaného sousedství, čímž se sníží původních 12 hodnot (pozice obou bodů a jejich normály) na 4. Pro vytvoření konečné PFH reprezentace pro daný bod je množina všech čtveřic sestavena do histogramu. To uděláme tak, že každou jednotlivou hodnotu z dané čtveřice rozdělíme na b pod-intervalů (*Bins*) a spočítáme počet výskytů v těchto intervalech. Vzhledem k tomu, že jsou 3 hodnoty z této čtveřice úhly mezi normálami, mohou být jejich hodnoty jednoduše normalizované do shodného intervalu jednotkové kružnice. V některých případech, čtvrtý prvek d, nepřináší velký přínos. Naopak, když vezmeme v úvahu mračna bodů získané v robotice, kde se vzrůstající vzdáleností od senzoru roste i vzdálenost mezi jednotlivými body, tak vynechání tohoto prvku je spíše přínosné.

V PCL výchozí implementace PFH používá rozdělení na 5 intervalů a nezahrnuje vzdálenosti mezi body. Z toho plyne že velikost histogramu má 125 hodnot (5^3). Časová složitost algoritmu je $O(nk^2)$.

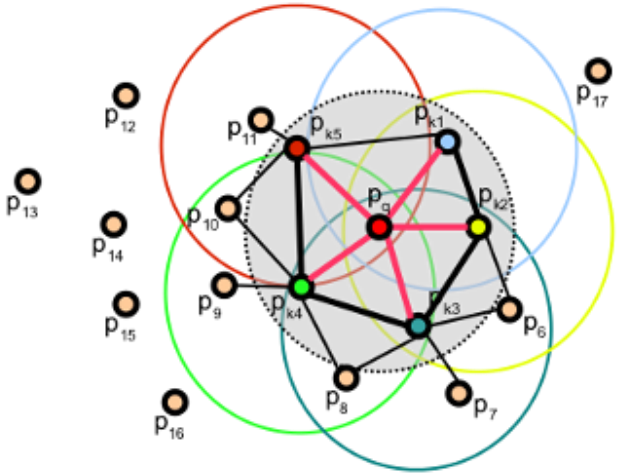
2.6.2 FPFH

Jedná se o zjednodušení PFH, a proto se nazývá Fast Point Feature Histogram (18). Snižuje výpočetní složitost algoritmu na $O(nk)$, přičemž si stále zachovává většinu popisné síly PFH.

V prvním kroku je pro každý bod p_q vypočítaná trojice $\langle \alpha, \varphi, \theta \rangle$ mezi ním a jeho k-sousedy stejným způsobem, jak je popsáno v kapitole 2.6.1. Hovoříme zde o simplified PFH neboli SPFH. V dalším kroku se vypočítá hodnota FPFH pro každý bod p_q pomocí SPFH a jeho sousedů dle vzorce:

$$FPFH(p_q) = SPFH(p_q) + \frac{1}{k}\sum_{i=1}^{k}\frac{1}{\omega_k} \cdot SPFH(p_k) \tag{24}$$

Kde váha ω_k reprezentuje vzdálenost mezi dotazovaným bodem p_q a sousedním bodem p_k v určitém zvoleném metrickém prostoru, čímž se získá pár (p_q, p_k). Na obrázku níže je zobrazen diagram vztahů pro okolí bodu p_q.



Obrázek 12: Digram vztahů bodů z okolí bodu p_q pro výpočet deskriptoru FPFH.

Hlavní rozdíly mezi PFH a FPFH:

- FPFH plně nespojuje všechny sousedy p_q, a proto můžou chybět některé páry, které by jinak přispěly k zachycení přesnější geometrie kolem dotazovaného bodu.
- PFH modeluje přesně ohraničený povrch okolo bodu, zatímco FPFH zahrnuje i další páry mimo poloměr r (nejvýše 2r)
- Kvůli opětovnému vážení kombinuje FPFH hodnoty s SPFH a znovu obnovuje některé sousední páry.
- Celková složitost FPFH je značně snížena, což umožňuje jeho využití v aplikacích v reálném čase.
- Výsledný histogram FPFH se zjednoduší odstraněním souvztažností mezi hodnotami. Jednoduše se vytvoří histogram pro každou hodnotu a následně se zřetězí (*Concatenation*).

V PCL výchozí implementace FPFH používá rozdělení na 11 intervalů. Z toho plyne že velikost histogramu má 33 hodnot (11×3). Příklad, jak použít FPFH v knihovně PCL je znázorněn viz Zdrojový kód 8. Na vstupu předáváme metodě mračno bodů *cloud*, normály *normals* přidružené k těmto bodům,

vybrané klíčové body *keypoints*, pro které chceme deskriptory počítat a poloměr *RADIUS* ve kterém budeme počítat se sousedními body.

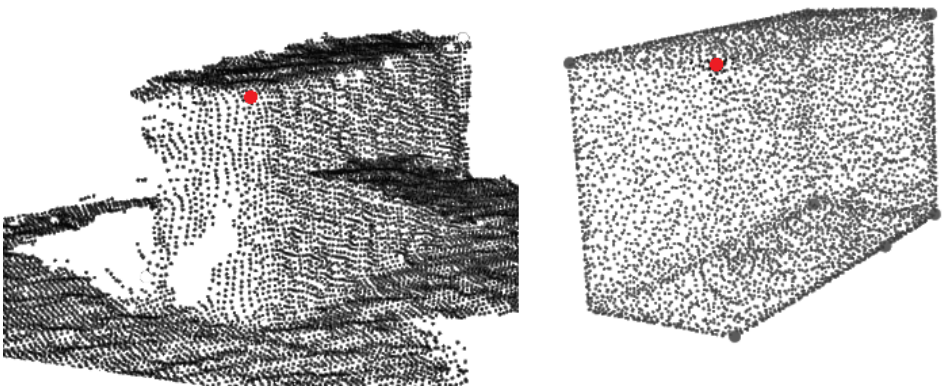
```
pcl::FPFHEstimation<PointType, NormalType, FPFHSignature33> fpfh;
fpfh.setSearchSurface(cloud);
fpfh.setInputNormals(normals);
fpfh.setInputCloud(keypoints);
pcl::search::KdTree<PointType>::Ptr tree(new pcl::search::KdTree<PointType>);
fpfh.setSearchMethod(tree);
fpfh.setRadiusSearch(RADIUS);
fpfh.compute(*outDescriptors);
```

Zdrojový kód 8: Použití FPFH v knihovně PCL.

2.7 Korespondující body

Poté, co jsou vypočítány deskriptory pro aktuální scénu, a každý hledaný model, následuje hledání všech korespondujících párů (tj. bod ze scény + bod z modelu). Nejjednodušším způsobem, jak určit, zda se dva body shodují, je spočítat euklidovskou vzdálenost mezi jejich deskriptory. Pokud by tato hodnota byla menší než určitý práh, pak můžeme body prohlásit za korespondující. Výhoda tohoto přístupu je, že hledáme body jejichž vzdálenost je co nejmenší, a k tomu se dá využít akcelerační datová struktura k-d strom. Nevýhoda je, že současné senzory ještě nejsou natolik přesné, aby to, podle mého názoru, mohlo dobře fungovat.

V reálné situaci zjišťujeme, že dva body, které mají korespondovat, nejsou ve skutečnosti úplně totožné. Ať už za to může nepřesnost senzoru, nebo nedokonalost reálných předmětů. Podle mě mohou být dva body absolutně totožné pouze v jediné situaci a to, když porovnáváme syntetický model se syntetickou scénou. To ale není to, co chceme. V našem případě potřebujeme porovnat „dokonalý“ klíčový bod z modelu s bodem ze scény, který je pravděpodobně zašuměný nebo i neúplný (viz Obrázek 13). Jeden způsob, jak se s tím vypořádat by mohlo být zmenšení počtu hodnot, které nám udává deskriptor a tím se zbavit nadbytečných detailů, které nám v dané situaci spíše ubližují. Potom zde máme ještě druhý způsob, kterým jsem se rozhodl vydat, a to s využitím neuronových sítí (popsáno viz kapitola 3).



Obrázek 13: Objekt, jak ho vidí senzor (vlevo) vs synteticky vytvořený model (vpravo).

2.8 Geometrická konzistence (19)

Nyní už máme k dispozici seznam korespondencí mezi klíčovými body z modelů a ze scény. To ale ještě neznamená, že se tam dané objekty nacházejí. Například mějme nalezené dvě korespondence pro daný objekt. Jaká by měla být vzdálenost mezi těmito body je nám známo z modelu. Podíváme se tedy na jejich vzdálenost i ve scéně. Jestliže zjistíme, že se vzdálenosti mezi sebou liší, tak tyto body do daného objektu pravděpodobně nepatří. Takovou kontrolu implementuje krok zvaný shlukování korespondencí. Musíme taky počítat s tím, že pro získání 6 DoF pozice objektu potřebujeme alespoň 3 body. Podle složitosti modelu je lepší mít bodů více např. 5–10.

Mějme list korespondujících párů $L = \{C_1, C_2, \dots, C_n\}$. Jeden pár obsahuje jeden klíčový bod ze scény a jeden z modelu $C = \{S, M\}$. Pro každý pár z listu provedeme následující postup seskupování (budeme mít n skupin): Aktuální pár inicializuji jako první pár ve skupině. Do skupiny dále přiřadíme všechny ostatní páry, které splňují následující podmínku pro C_1 *a* C_2:

$$|d(S_1, S_2) - d(M_1, M_2)| < \epsilon \tag{25}$$

Kde $d(S_1, S_2)$ je euklidovská vzdálenost mezi dvěma klíčovými body ve scéně a obdobně pak $d(M_1, M_2)$ je euklidovská vzdálenost mezi dvěma klíčovými body v modelu. ϵ je prahová hodnota, která rozhoduje jestli C_1 *a* C_2 splňují geometrické omezení nebo ne. Největší vytvořená skupina má největší pravděpodobnost, že právě její korespondence jsou ty skutečné. Pro ověření se odhadne tuhá transformace (*Rigid Transformation*) použitím reprezentace kvaternionů (*Quaternion*) (20) a ta se následně ohodnotí s přihlédnutím na vzdálenosti jednotlivých bodů transformace od bodů ze scény.

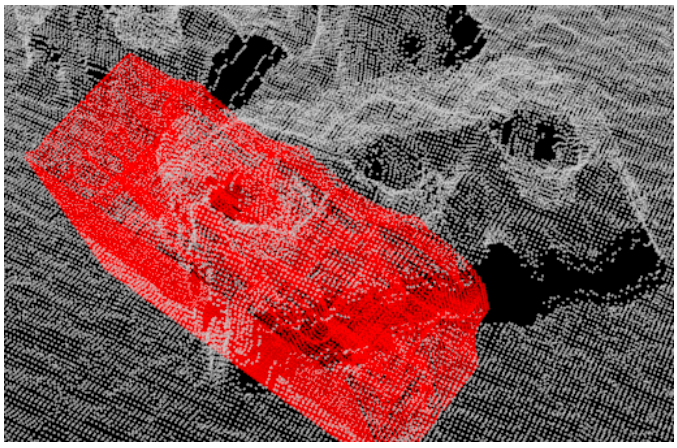
Použití v knihovně PCL vypadá následovně: viz Zdrojový kód 9. Parametr *GC_SIZE* odpovídá parametru ϵ viz vysvětlení výše. Parametr *GC_THRESH* definuje nejmenší počet korespondencí, které může výsledná podmnožina obsahovat.

```
pcl::GeometricConsistencyGrouping<PointType, PointType> gc_clusterer;
gc_clusterer.setGCSize(GC_SIZE);
gc_clusterer.setGCThreshold(GC_THRESH);
gc_clusterer.setInputCloud(modelKeypoints);
gc_clusterer.setSceneCloud(sceneKeypoints);
gc_clusterer.setModelSceneCorrespondences(Corrs);
gc_clusterer.recognize(*outRototrs, *outClusteredCorrs);
```

Zdrojový kód 9: Použití geometrické konzistence v knihovně PCL.

V praxi, v závislosti na parametrech, můžeme touto metodou pro jeden model získat i několik transformačních matic, které udávají, kde ve scéně by se mohl hledaný objekt nacházet. Většina transformací bude pravděpodobně chybná, např. vlivem příliš velkého množství nalezených korespondencí v přeplněné scéně, nebo i velkou podobností jednotlivých modelů. Také se může stát, že máme ve scéně stejný objekt zastoupený vícekrát. Získáme tedy více validních korespondencí a Geometrická konzistence může začít nacházet i takové transformace, které „visí“ mezi těmito objekty. Na druhou stranu se taky může stát, že žádná nalezená transformace nebude správná.

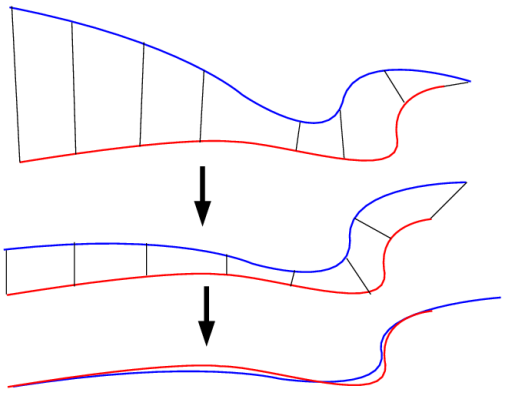
Když si vizualizujeme odhadnuté transformace. Může se stát, že budou některé objekty vyčnívat ze scény do prostoru (viz Obrázek 14). Způsob, který používám pro odfiltrování těchto transformací, je takový, že se u každé transformace podívám na klíčové body, které k transformaci vedli, a porovnám jejich normály s příslušnými normálami ve scéně pomocí skalárního součinu (*Dot product*). Pokud je průměrný skalární součin menší než stanovený práh 0.45, tak transformaci zahodím. Zároveň pokud je počet klíčových bodů podílejících se na transformaci menší než 4, tak transformaci zahodím. Po odfiltrování těchto transformací jsem ještě schopen ty zbylé seřadit podle počtu bodů, které do transformace přispěly, a velikosti průměrné odchylky jejich normál.



Obrázek 14: Ukázka špatné transformace po geometrické konzistenci.

2.9 Iterative Closest Point

Úkolem algoritmu ICP je minimalizovat vzdálenost mezi dvěma množinami bodů. Finální odhad 6DoF pózy je tedy získán určením rotace a translace bodů hledaného objektu tak, aby jejich vzdálenost od bodů prohledávané scény byla co nejmenší. Podmínkou pro úspěšné nalezení finální transformace je dostatečně podobný prvotní odhad, jinak může dojít k uváznutí vzdálenosti mezi množinami bodů v lokálním minimu. Jedním způsobem je nalezení nejbližšího bodu v druhé množině, což je obvykle robustní a stabilní řešení pro tělesa se složitými geometrickými tvary. Algoritmus probíhá rekurzivně, dokud není dosažena podmínka pro zastavení, např. dosažení vzdálenosti menší než zadaná prahová hodnota nebo provedení určeného počtu iterací.



Obrázek 15: Ukázka zarovnání dvou křivek pomocí ICP.

Algoritmus ICP je implementován v knihovně PCL. Jeho použití je znázorněno v následujícím zdrojovém kódu:

```
pcl::IterativeClosestPointWithNormals<pcl::PointNormal, pcl::PointNormal> icp;
icp.setInputSource(model);
icp.setInputTarget(scene);
icp.setMaxCorrespondenceDistance(MAX_CORR_DISTANCE);
icp.align(*finalCloud);
Eigen::Matrix4f transformation = icp.getFinalTransformation();
```

Zdrojový kód 10: Použití ICP v knihovně PCL.

Prvním krokem po vytvoření detektoru je nastavení vstupního a cílového cloudu pomocí *setInputCloud* a *setTargetCloud*, kde rotace probíhá na vstupním cloudu tak, aby se minimalizovala vzdálenost korespondujících bodů mezi cloudy. Dále lze nastavit nejrůznější parametry, např. maximální vzdálenost mezi korespondujícími body *MAX_CORR_DISTANCE*, maximální počet iterací a minimální hodnotu epsilon (velikost změny mezi iteracemi). Transformace se spouští metodou *align*, jejíž výstupem je výsledný point cloud *finalCloud*. Transformační matici lze získat pomocí *getFinalTransformation*.

Nyní, když máme náš objekt zarovnaný na jeho finální pozici, pokusíme se pro něj vypočítat registrační skóre. Dalo by se říct, míru toho jak dobře finální objekt do scény „zapadne“. K tomuto účelu jsem si napsal vlastní funkci, která bere v potaz množství „dobrých a špatných“ bodů finálního objektu vůči bodům ze scény a celkovou velikost objektu. Velikost registračního skóre se řídí podle tohoto navrženého vzorce:

$$score = \left(\frac{ok}{all} - \frac{3 \times bad}{bad+ok}\right) \times \sqrt{all} \tag{26}$$

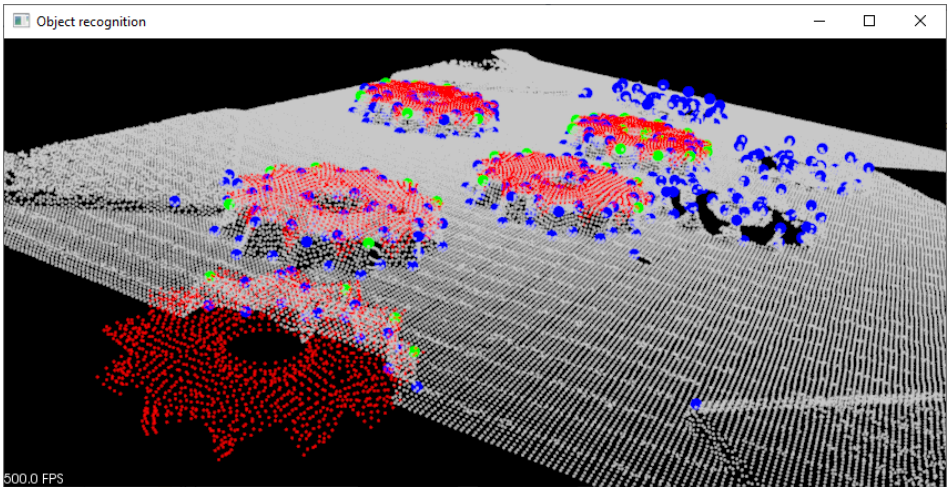
Kde ok je počet bodů finálního objektu, jejichž vzdálenost od nejbližšího bodu ze scény je menší než $MAX\_CORR\_DISTANCE$. bad je počet bodů finálního objektu, jejichž vzdálenost od nejbližšího bodu ze scény je větší než $MAX\_CORR\_DISTANCE$ a zároveň je menší než $2.5 \times MAX\_CORR\_DISTANCE$. A kde all je celkový počet bodů finálního objektu.

Myšlenka je taková, že čím vyššího skóre finální objekt dosáhne tím lépe pasuje do prohledávané scény. Pokud je hodnota skóre menší než 0, tak považujeme transformaci finálního objektu za špatnou.

2.10 Vizualizace

PCL obsahuje vizualizační modul založený na knihovně VTK (*Visualization Toolkit*) (21). Tato knihovna obsahuje metody pro vykreslování mračen bodů ve formátu pcl::PointCloud<T> s možností nastavit některé jejich vizuální vlastnosti jako například barvy, velikosti bodů, normály, průhlednost nebo geometrii. Umožňuje vykreslovat základní 3D tvary jako jsou válce, koule, čáry nebo mnohoúhelníky. Obsahuje i modul pro vizualizaci 2D grafů. Ve vizualizačním okně můžeme s mračnem manipulovat pomocí myši nebo klávesnice a je zde i možnost registrovat vlastní chování na základě stisku klávesnice (*KeyboardEvent*) nebo kliknutí na bod (*PointPickEvent*).

V aplikaci vykresluji scénu bílou barvou a nalezené instance objektů červenou barvou. Nalezené klíčové body jsou ve scéně vykresleny modře a korespondující klíčové body, které dopomohly k nalezení transformace zeleně. Ukázka vizualizačního okna viz Obrázek 16.



Obrázek 16: Vizualizace pomocí knihovny VTK. Scéna z datasetu ITODD.

Následující zdrojový kód ukazuje, jak toto okno vytvořit a zobrazit v něm požadované mračno bodů a spolu s ním i klíčové body, jejichž velikost nastavíme na 10 aby byli zobrazeny větší.

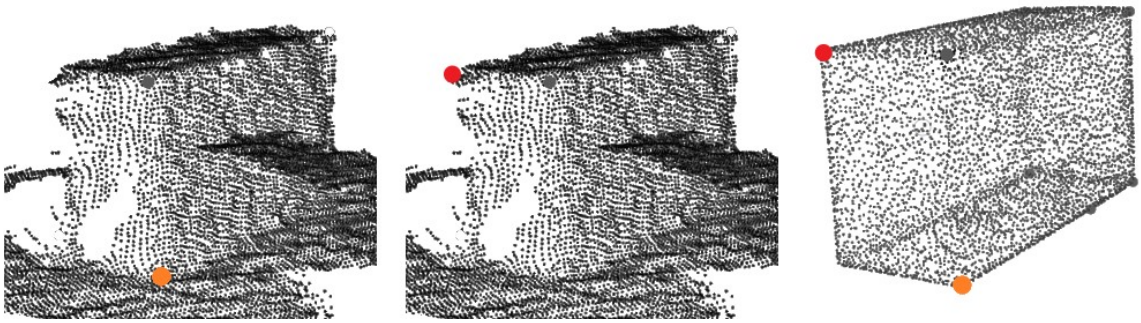
```
pcl::visualization::PCLVisualizer viewer("Viewer_name");
viewer.addPointCloud(cloud, "cloud_name");
viewer.addPointCloud(keypoints, "keypoints");
viewer.setPointCloudRenderingProperties(
       pcl::visualization::PCL_VISUALIZER_POINT_SIZE, 10, "keypoints"
);
viewer.spinOnce();
```

Zdrojový kód 11: Zobrazení mračna bodů a klíčových bodů pomocí modulu VTK v knihovně PCL.

3 Hledání korespondencí pomocí neuronové sítě

Hlavní myšlenkou tohoto přístupu je, že budeme porovnávat každý klíčový bod z modelu s každým klíčovým bodem ze scény pomocí neuronové sítě. Velikost vstupní vrstvy (*Input Layer*) se tedy odvíjí od velikosti vybraného deskriptoru. NS poté vyhodnotí, zda jsou dané dva body korespondující nebo ne. Pro tento typ rozpoznávání by nám měla stačit, relativně jednoduchá, plně propojená (*Fully-Connected*) neuronová síť. Předpokládám jednu nebo dvě vnitřní vrstvy (*Hidden Layers*), každá cca o velikosti použitého deskriptoru (více popsáno viz kapitola 3.1). Podle mého názoru by tímto přístupem měl odpadnout problém, že si jednotlivé korespondující body ve skutečnosti nejsou úplně podobné. A pokud bude NS dostatečně dobře naučená a dokáže eliminovat co nejvíce tzv. False Positive, mělo by to i razantně zvýšit celkovou rychlost rozpoznávání.

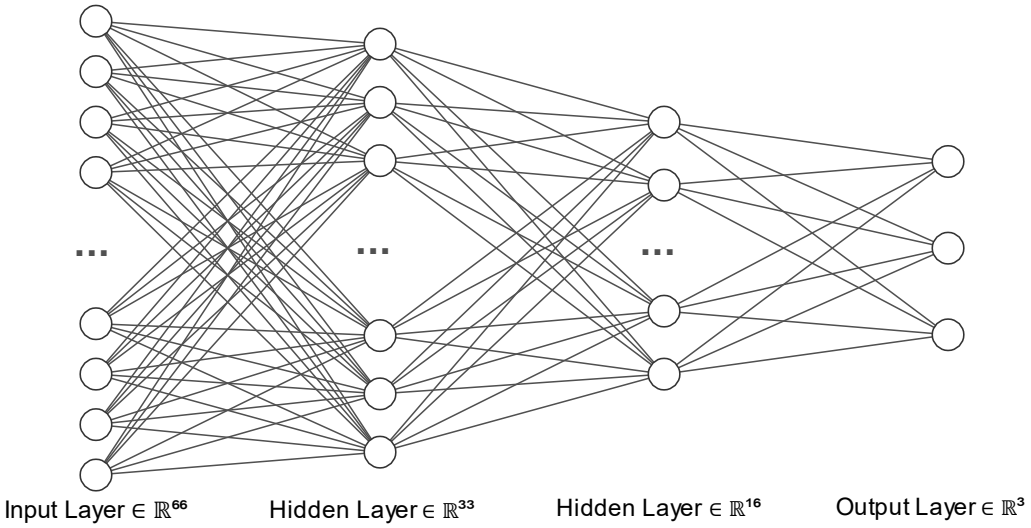
Teoreticky by NS mohla být schopná správně vyhodnotit i situace, kde nám pro daný klíčový bod chybí některé okolní body. Například z důvodu, že na ně senzor nevidí (viz Obrázek 17 – uprostřed). Dokonce by to mohlo vyřešit i situaci, kdy je objekt v kontaktu s jiným objektem (např. leží na podlaze) a klíčový bod má ve svém okolí body, které nenáleží hledanému objektu (viz Obrázek 17 – vlevo). Abych mohl tuto hypotézu ověřit, rozhodl jsem se tyto body v NS rozpoznávat jako „poloviční korespondence“ a otestovat jejich vliv na celkový výkon rozpoznávání.



Obrázek 17: Vlevo – Klíčový bod leží na rozhraní objekt-podlaha. Uprostřed – Okolní body pro počítaný klíčový bod nejsou na zadní straně ve scéně vidět.

3.1 Architektura sítě

Na vstupu NS předáme hodnoty obou deskriptorů (D_M, D_S) právě vyšetřovaných klíčových bodů, kde první patří modelu a druhý je ze scény. To znamená, že vstupní vrstva neuronové sítě bude nabývat velikosti $2 \times velikost\ použitého\ deskriptoru$. Já jsem si vybral v aplikaci používat deskriptor FPFH (viz kapitola 2.6), který má velikost 33. Velikost vstupní vrstvy bude tedy 66. Na výstupu NS potřebujeme, aby výstupní vrstva (*Output Layer*) byla schopná klasifikovat tyto případy: Body **nejsou korespondující**, body **jsou korespondující** a body **jsou napůl korespondující** (viz Obrázek 17). Velikost výstupní vrstvy bude tedy nabývat velikosti 3. Ještě je potřeba určit velikost a počet skrytých vrstev. Vzhledem k tomu, že neexistuje jednoznačný způsob, jak by měly tyto velikosti být správně navoleny. Tak jsem se rozhodl začít se dvěma skrytými vrstvami, první o velikosti 33 a druhá o velikosti 16. Jako aktivační funkci používám Relu. Později můžu zkusit jiné konfigurace a experimentálně ověřit, která podává nejlepší výkon. Níže je znázorněný diagram NS (viz Obrázek 18) a ukázka, jak je tato architektura definována v kódu pomocí knihovny Dlib (viz Zdrojový kód 12).



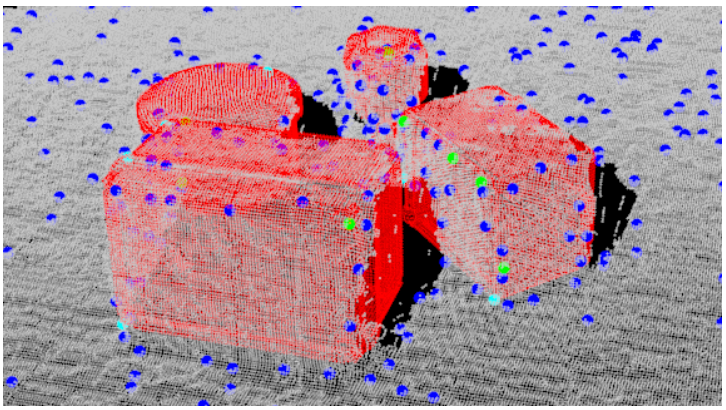
Obrázek 18: Zjednodušený diagram použité neuronové sítě.

```
using neuralNet = dlib::loss_multiclass_log<
        dlib::fc<3,
        dlib::relu<dlib::fc<16,
        dlib::relu<dlib::fc<33,
        dlib::input<dlib::matrix<float>>
        >>>>>>;
```

Zdrojový kód 12: Definice architektury NS v knihovně Dlib.

3.2 Trénovací množina

Vzhledem k jednoduchosti sítě předpokládám, že nám pro natrénování bude stačit relativně malá trénovací množina s velikostí v řádech tisíců prvků, řekněme 20 tisíc. Aby nebylo nutné označovat veškeré korespondence ručně, pomůžeme si s anotovanými datasety z kapitoly 1.7 BOP: Benchmark for 6D Object Pose Estimation. Datasety vždy obsahují scénu a správně umístěné instance objektů, které se v ní nacházejí (viz Obrázek 19). Abychom toho mohli využít, musíme nejprve scénu zpracovat, stejně jako kdybychom v ní chtěli rozpoznávat, tedy veškeré kroky od kapitoly 2.2 po 2.6. Když máme veškeré klíčové body i jejich deskriptory připravené, můžeme začít tvořit trénovací množinu.



Obrázek 19: Správné umístění objektů ve scéně podle BOP. Modré body – veškeré klíčové body nalezené ve scéně. Zelené body – klíčové body jejichž pozice se u modelu i scény shoduje.

3.2.1 Automatizovaný výběr z překrývajících se klíčových bodů

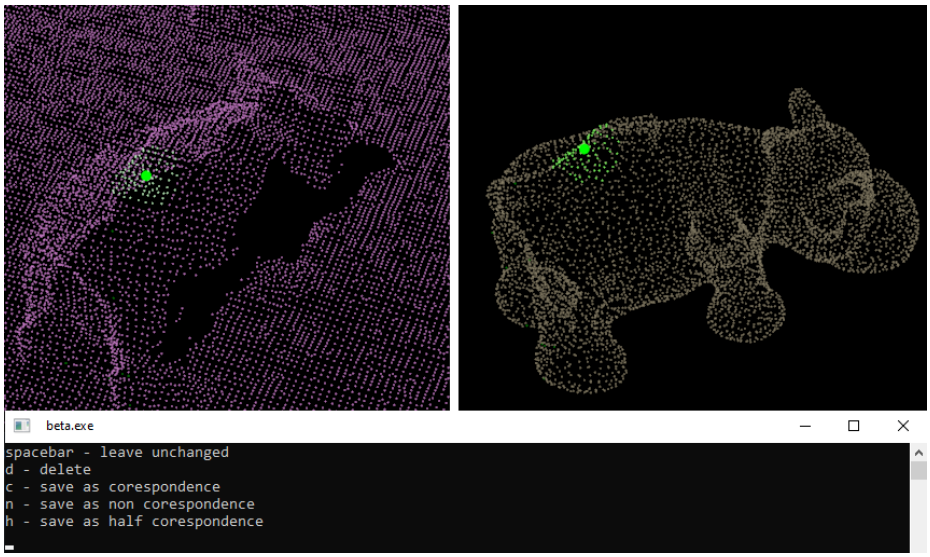
Začneme automatizovaným výběrem všech korespondujících párů, jejichž klíčové body umístěných modelů se překrývají s klíčovými body ve scéně (viz Obrázek 19), tedy jejichž vzdálenost je menší než stanovený práh. Z klíčových bodů, které se nepřekrývají si poté náhodně některé vybereme a označíme je jako nekorespondující. Získané korespondující i nekorespondující páry si uložíme do CSV souboru, kde každý pár obsahuje informace o scéně, modelu a konkrétních klíčových bodech, ze kterých byl vytvořen. Data v tomto formátu nám zajistí opakované použití stejných klíčových bodů pro výpočet trénovacích množin. To je důležité hlavně v momentě, kdy budou korespondence vybrány ručně.

3.2.2 Automatizovaný výběr ze syntetických modelů

Druhý automatizovaný způsob výběru korespondencí je proveden pouze na modelech. Určení korespondenčních párů je zde jednoduché. Pro každý nalezený klíčový bod řekneme, že je korespondující sám k sobě. Při hledání nekorespondujících párů budeme náhodně tvořit dvojice klíčových bodů a pokud bude jejich euklidovská vzdálenost větší než určitý práh, prohlásíme je za nekorespondující.

3.2.3 Manuální výběr

Automatický výběr korespondencí nám ušetří spoustu času, ale je zde reálné riziko, že ne všechny korespondence budou určeny správně. V tomto kroku se tedy pokusím veškerá data projít a vybrat z nich pouze ty, které podlé mého subjektivního názoru považuji správná. Abych mohl rozhodnout, zdali budu považovat dva body za korespondující, musím si je nejprve vykreslit na obrazovku. V jednom okně bude vykresleno mračno bodů ze scény a v druhém okně mračno bodů z modelu. V těchto mračnech bude zvýrazněn vyšetřovaný klíčový bod i s jeho okolím (viz Obrázek 20). Pohledem na tyto body poté rozhodnu, zdali je uložím jako korespondující, nekorespondující, napůl korespondující anebo je smažu. Validované korespondující páry poté uložím do nového CSV souboru stejným způsobem jako ty automaticky získané.



Obrázek 20: Vizualizace klíčových bodů mezi scénou a modelem pro manuální potvrzení korespondence. Poznámka: Tyto body by byly uznány jako korespondující.

3.2.4 Formát trénovací množiny

Máme-li vybrané veškeré korespondence, můžeme z nich začít vytvářet trénovací množinu. Pro každý klíčový bod vypočítáme jeho deskriptor. Pro každý pár tak získáme dva deskriptory, které uložíme do CSV souboru ve tvaru: $D_M, D_S, Output$. D_M je deskriptor klíčového bodu z modelu. D_S je deskriptor klíčového bodu ze scény a $Output$ označuje o jaký druh korespondence se jedná (0 – nejsou korespondující, 1 – jsou korespondující, 2 – jsou napůl korespondující). V případě automatického výběru bude $Output$ nabývat pouze hodnot 0 a 1.

3.3 Učení sítě

Pomocí získané trénovací množiny můžeme nyní naši síť začít učit. K tomu nám v Dlib poslouží dnn_trainer. Nastavíme několik parametrů: **Míra učení** (*Learning Rate*) – určuje velikost změny vah v každé iteraci směrem k minimu ztrátové funkce (*Loss Function*). **Velikost šarže** (*Batch Size*) – počet trénovacích vzorků zpracovaných v jedné iteraci. Maximální **počet epoch**. Je vhodné zamíchat (*Shuffle*) vstupní data z trénovací množiny, aby se síť učila rovnoměrně. K tomu nám poslouží funkce `randomize_samples`. Poté síť začneme trénovat. Trénování je ukončeno po dosažení maximálního počtu epoch nebo když `learning_rate` klesne pod hodnotu stanovenou pomocí `min_learning_rate`. Naučenou síť poté vyčistíme od dočasných dat a uložíme. Ukázka, jak vypadá trénování NS v knihovně Dlib:

```
neuralNet net;
dlib::dnn_trainer<neuralNet> trainer(net);
trainer.set_learning_rate(0.1);
trainer.set_min_learning_rate(0.0001);
trainer.set_mini_batch_size(1024);
trainer.set_max_num_epochs(3000);
dlib::randomize_samples(training_inputs, training_labels);
trainer.train(training_inputs, training_labels);
net.clean();
dlib::serialize("neuralNet.dat") << net;
```

Zdrojový kód 13: Trénování NS v knihovně Dlib.

3.4 Použití sítě

Pro použití sítě nám stačí danou síť načíst a předat jí vstupní data, která chceme klasifikovat. Vstupní data předáváme ve formátu: D_M, D_S (viz kapitola 3.1). Přičemž každý deskriptor D_M je v relaci s každým deskriptorem D_S. Jednotlivé relace uložíme do datové struktury dlib::matrix<float>, se kterou umí neuronová síť pracovat. Máme-li připravené veškeré vstupy, předáme je neuronové síti a počkáme na odpověď. Ukázka použití v kódu:

```
neuralNet net;
dlib::deserialize("neuralNet.dat") >> net;
std::vector<dlib::matrix<float>> inputs = GetInputs();
std::vector<unsigned long> outputs = net(inputs);
```

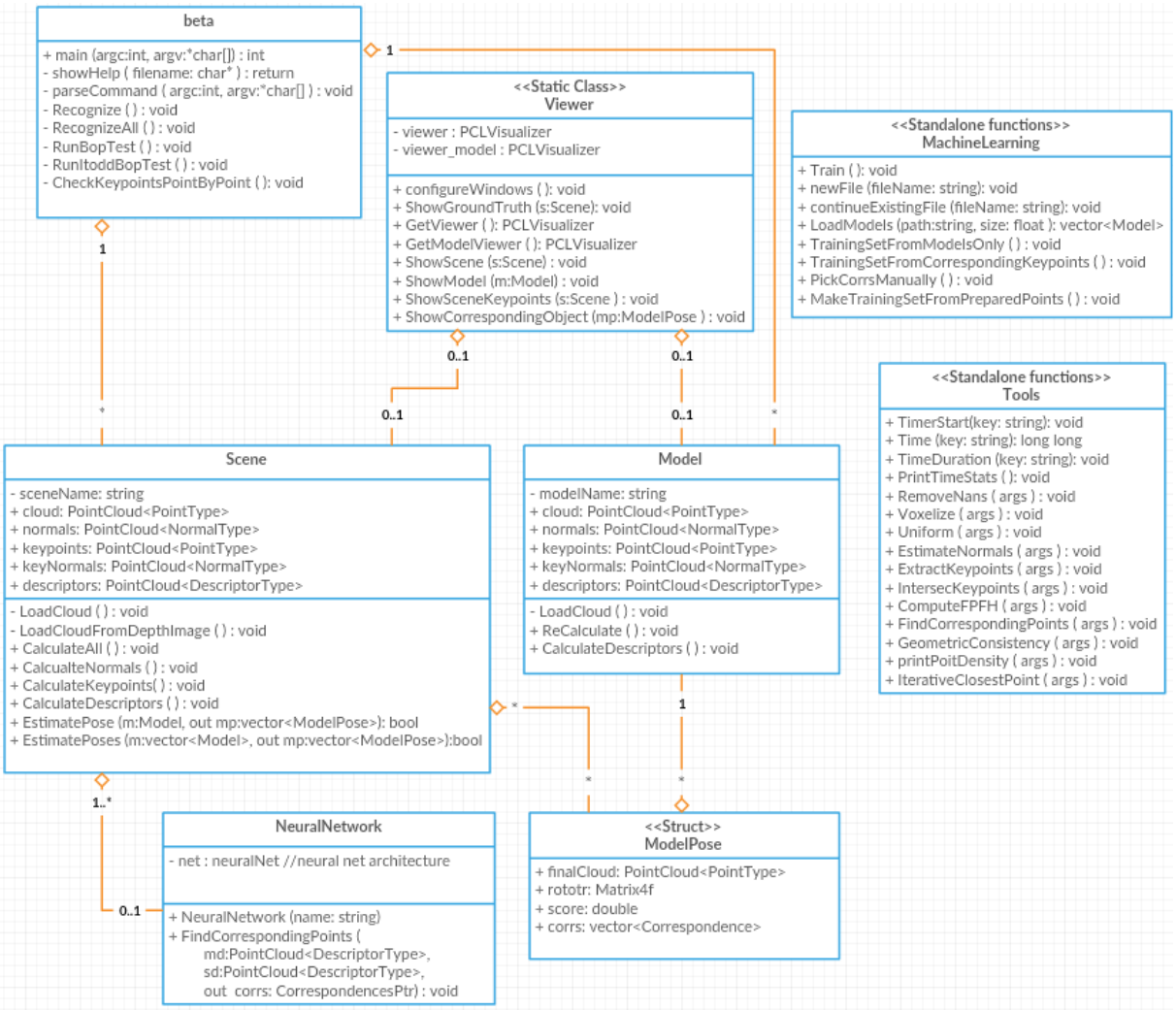
Zdrojový kód 14: Použití NS v knihovně Dlib.

4 Aplikace

V této kapitole se podíváme na to, jakým způsobem jsem aplikaci implementoval a ukážeme si, jak funguje. Aplikaci je napsaná ve Visual Studiu 2017 v jazyce C++.

4.1 Třídní diagram

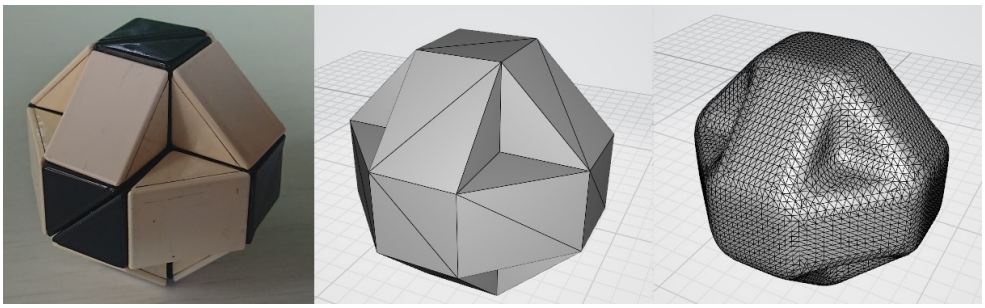
Z třídního diagramu (viz Obrázek 21) je vidět zvolená architektura aplikace. Aplikace je rozdělená do několika souborů (.cpp a .h), které odpovídají objektům z diagramu s výjimkou struktury *ModelPose*, která je definována ve třídě *Scene.cpp*. Hlavní metoda *main*, která je volána z příkazové řádky je obsažena v souboru *beta.cpp*. U některých funkcí, kterým se předává větší množství parametrů, jsem pro zachování čitelnosti, jejich parametry nevypisoval viz *Tools* v diagramu. Kroky rozpoznávacího řetězce popsané v kapitole 2 jsou implementované v třídě *Tools* a jsou volány z třídy *Scene* popřípadě *Model*. Neuronová síť, která svou funkčností nahrazuje funkci *FindCorrespondingPoints* z třídy *Tools* je implementována ve třídě *NeuralNetwork*. Třída *MachineLearning* obsahuje metody k vytvoření trénovací množiny a k samotnému trénování neuronové sítě (více popsáno v kapitole 3).



Obrázek 21: Třídní diagram aplikace.

4.2 Použití – standartní režim

Nyní se podíváme na typický případ použití, kdy máme k dispozici reálný objekt a jeho 3D CAD model. Nejprve, jak je popsáno v kapitole 2.1, si model předzpracujeme. K tomuto účelu jsem si vytvořil samostatnou konzolovou aplikaci, která pracuje jak s aplikací MeshLab tak s aplikací Instant Meshes. Její zdrojové kódy jsou dostupné v elektronické příloze. Ukázka objektu a jeho modelu je vidět na následujícím obrázku:



Obrázek 22: Vlevo – reálný objekt. Uprostřed – CAD model. Vpravo – Remesh modelu.

Pomocí hloubkového senzoru (RealSense v mém případě) si vytvořím snímek hloubkové mapy a uložím si ho například ve formátu PLY. Ukázka, jak může získaná taková scéna vypadat:

Obrázek 23: Vlevo – RGB obraz scény. Vpravo – 3D snímek scény.

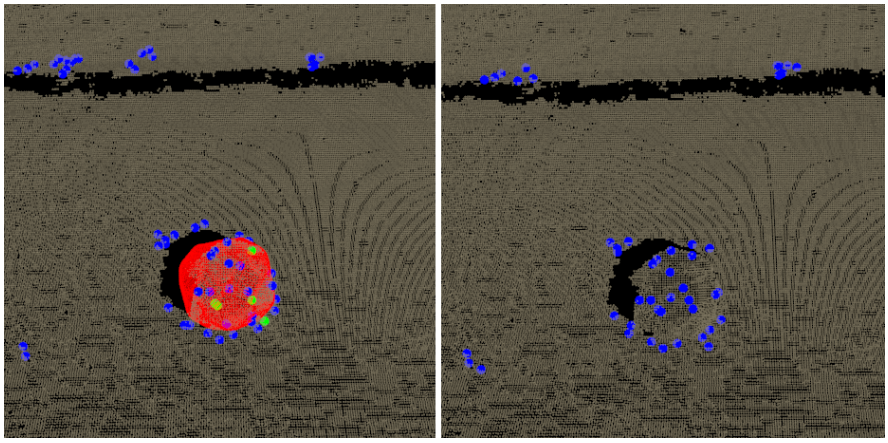
Aplikaci nyní spustíme tímto příkazem (detaily používání aplikace jsou popsány v příloze):

- beta.exe MODELS SCENES NN_NAME [Options]

Konkrétní příkaz v tomto je v tomto ukázkovém případě:

- beta.exe Realsense\models_ready Realsense\scenes\1.ply Real --detailScale 0.66 --sceneScale 1000

Od této chvíle se spustí rozpoznávací řetězec popsaný v kapitole 2, konkrétně kroky od načtení (viz kapitola 2.2) až po vizualizaci (viz kapitola 2.10). Na konci běhu programu by se nám tak na obrazovku měl vykreslit daný model umístěný na patřičnou pozici ve scéně (viz Obrázek 24). Úspěšnost nalezení správné pozice je v tomto případě kolem ~50%. Příčinou toho, že se objekt nepodaří najít, může být, podle mého názoru, nedostatek klíčových bodů nebo špatná kvalita nasnímané scény.



Obrázek 24: Vlevo – situace kdy je model korektně rozpoznán a umístěn. Vpravo – situace kdy se model nepodařilo rozpoznat.

4.3 Použití – trénovací režim

Kromě standartního rozpoznávacího režimu má aplikace i druhý tzv. trénovací režim, který využívá metody z třídy *MachineLearning* (viz Obrázek 21). Režim slouží pro vytváření trénovacích množin (viz kapitola 3.2), trénování neuronové sítě (viz kapitola 3.3) a testování rozpoznávacího výkonu aplikace, které budeme potřebovat v následujících experimentech.

4.4 Výchozí parametry

V následující tabulce jsou vypsány výchozí parametry aplikace, ke kterým jsem postupně dospěl v průběhu vytváření aplikace.

Tabulka 2: Výchozí parametry aplikace.

Název parametru	Použití ve funkci	Hodnota	Poznámka
radiusNormal	EstimateNormals	4 mm	Viz Odhad normál v knihovně PCL.
radiusDescript	ComputeFPFH	12 mm	Viz Použití FPFH v knihovně PCL.
radiusOutlier	RemoveNaNs	6 mm	Viz Odstranění odlehlých bodů v knihovně PCL.
radiusKeypoint	ExtractKeypoints, IntersecKeypoints	6 mm	Viz Harris 3D v knihovně PCL.
tresholdHarris	ExtractKeypoints	0.001	
tresholdHarrisModel	ExtractKeypoints	0.0025	
GcToleratedDistance	GeometricConsistency	10 mm	Viz Použití geometrické konzistence v knihovně PCL.
voxelSize	Voxelize, Uniform	2.5 mm	Viz Voxelizace mračna v knihovně PCL.
detailScale	-	1	Koeficient pro stanovení úrovně hledaných detailů. Touto hodnotou vynásobíme všechny parametry jejichž hodnota je stanovena v milimetrech.

5 Experimenty

V této kapitole se podíváme na to, jak dobře se aplikaci daří rozpoznávat objekty ve scénách. Pro otestování jsem si připravil výběr 15 scén z datasetu tless (viz Obrázek 25), kde každá scéna je zobrazena z 5 různých úhlů. Dohromady bude tedy testování probíhat na 75 scénách. Pro vyhodnocení správnosti rozpoznávání využiji metodiku hodnocení používanou v BOP Benchmarku (viz kapitola 1.7.2). Tímto způsobem získáme výsledek jako procento správně rozpoznaných objektů z celkového množství objektů ve scénách (*Recall*). Kromě toho také porovnáme časy průměrného rozpoznávání na jednu scénu. Testování rozdělím na několik částí, kde v každé části budu zkoumat vliv daného parametru při rozdílném nastavení.

Obrázek 25: 15 různých scén z datasetu tless použitých pro otestování aplikace.

Veškeré výpočty byly prováděny na notebooku Lenovo Legion Y530 s následujícími parametry:

- Procesor: Intel Core i5-8300H @ 2.3 GHz (8 jader).
- Grafická karta: NVIDIA GeForce GTX 1060.
- Operační paměť: 8 GB 2667 MHz.
- Operační systém: Windows 10

Většina operací probíhá paralelně na CPU, protože se mi pro knihovnu PCL nepodařilo zprovoznit výpočty na GPU. Jediné výpočty aplikované na GPU jsou ty v rámci knihovny Dlib, tedy v rámci neuronové sítě.

Pokud není v rámci testu napsáno jinak, jsou ve všech testech výchozí parametry nastaveny na jejich základní hodnotu viz kapitola 4.4. S výjimkou parametru *detailScale*, který je nastaven na hodnotu 0.66. Rovněž se ve všech testech používá stejný testovací dataset *tless* popsaný výše.

5.1 Trénovací množina podle způsobu výběru

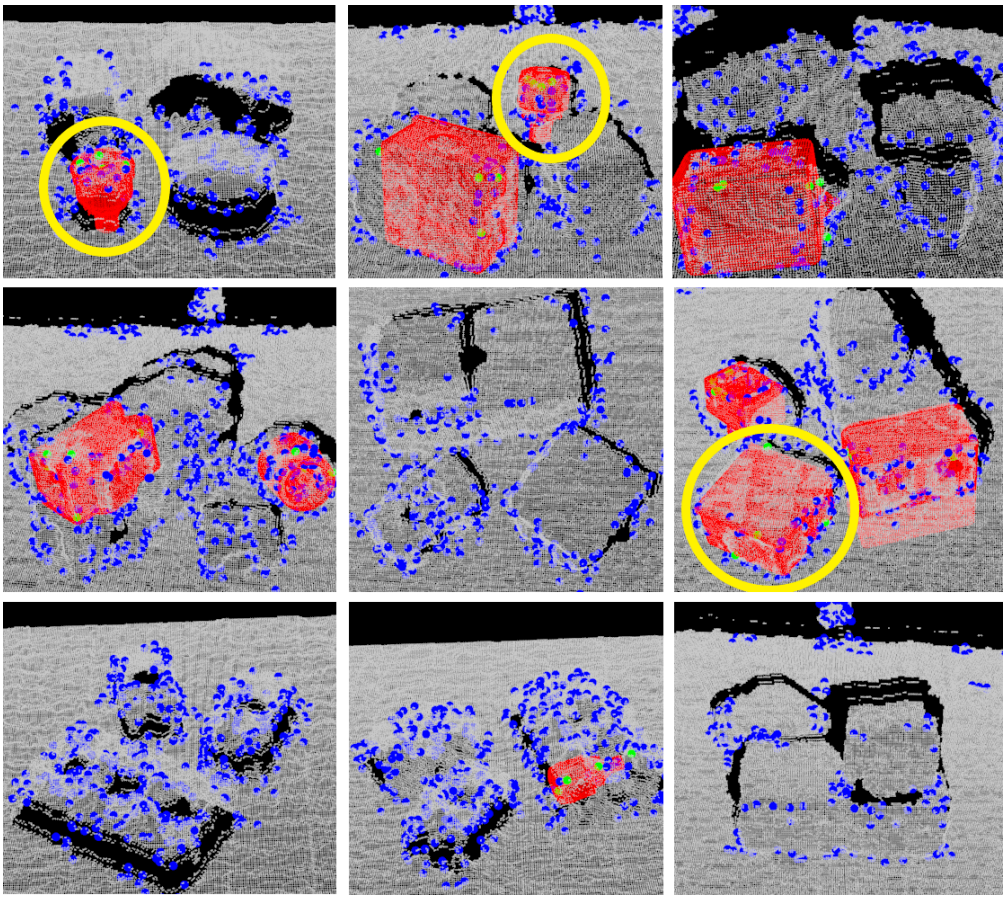
Zde zjišťuji, jaký vliv na výkon aplikace má způsob výběru trénovací množiny, která slouží pro natrénování neuronové sítě pro rozpoznávání korespondencí. Můžeme je rozdělit na množiny vytvořené čistě automatickou metodou (tj. Synt a Real) a množiny, které z nich vycházejí, ale byli ručně validovány bod za bodem (tj. Manual A a B). Postup získávání množin je blíže popsán v kapitole 3.2. Hlavními sledovanými parametry jsou zde velikost množiny a schopnost sítě se množinu naučit (*Training Accuracy*). Osobně očekávám, že nejlepších výsledků dosáhne aplikace při použití jedné z manuálně získaných trénovacích množin.

Tabulka 3: Výkon aplikace v závislosti na použité trénovací množině pro nacházení korespondencí.

Název	Popis	Velikost množiny (true; false)	Training accuracy	Recall	Průměrný čas na scénu [s]
Synt	Množina je automaticky získána pouze z modelů.	4106 (1360; 2746)	92.7%	**0.33%**	4.7
Real	Množina je automaticky získána ze scén a umístěných instancí modelů.	21577 (6830; 14747)	79.2%	**6.52%**	30.3
Manual A	Získána manuální validací množiny Real. A – pouze čisté korespondence. B – poloviční korespondence jsou taky brány jako korespondentní.	19882 (3536; 16346)	83.0%	3.67%	7.6
Manual B		19882 (4916; 14966)		**5.93%**	8.3
Man-Synt	Manual B + Synt	23988 (6276; 17712)	85.0%	4.31%	8.1

Nakonec si aplikace nejlépe vedla, co se rozpoznávání týče, s trénovací množinou „Real“. Hned za ní pak následovala množina „Manual B“. Nejhůře si pak vedla množina „Synt“, která v podstatě vyžaduje 100% schodu klíčových bodů, aby se její prvky dali považovali za korespondentní. Co se týče rychlosti rozpoznávání, tak manuálně validované množiny si vedli mnohem lépe než množina „Real“. Předpokládám, že je to způsobeno výrazným snížením počtu nalezených tzv. false positive korespondencí a tím pádem došlo k urychlení následujících kroků v rozpoznávacím řetězci.

Bohužel ani v nejlepším testovaném případě se zde nedá mluvit o dobré detekci. Částečně zde může být na vině fakt, že jsem se snažil rozpoznávání udělat univerzální pro co největší množství různých datasetů a testování jsem poté provedl pouze na datasetu tless, který považuji za velmi složitý (podobné objekty, velká míra šumu, vzájemný částečný překryv objektů). Přece jenom jsem doufal v o něco lepší výsledky. Na typický výsledek rozpoznávání při použití trénovací množiny Manual B se můžete podívat, viz Obrázek 26.



Obrázek 26: Typické výsledky rozpoznávání aplikace. Žlutě zakroužkované jsou korektní detekce.

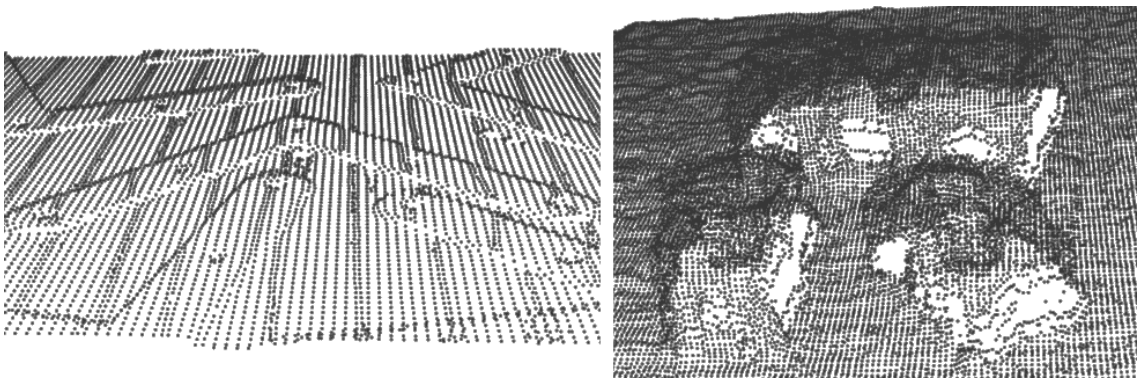
5.2 Trénovací množina podle druhu senzoru

Každý senzor má jiné vlastnosti, jiné rozlišení, přesnost, množství šumu nebo různé artefakty. Neuronové sítě zaměřené pouze na daný senzor by tak teoreticky měli při použití daného senzoru dosahovat lepších výsledků než obecně naučené neuronové sítě. V tomto experimentu zjistíme, jaký vliv na výkon aplikace mají trénovací množiny zaměřené na určitý druh senzoru. Použité trénovací množiny a senzory jsou vypsány v následující tabulce:

Tabulka 4: Trénovací množiny rozdělené podle použitého senzoru.

Senzor (Název trénovací množiny)	Dataset	Vlastnosti datasetu	Velikost množiny (true; false)	Training accuracy
Industrial	itodd	Použity velice přesné industriální senzory, scény jsou téměř bez šumu.	3257 (1208; 2049)	96.6%
PrimeSense	tless	Scény obsahují určité množství šumu.	5277 (1620; 3657)	89.4%
Mix	tless + itodd	-	8534 (2828; 5706)	83.0%

Obrázek 27 ukazuje rozdílnost mračen bodů pořízených zmíněnými senzory:



Obrázek 27: Vlevo – scéna pořízená industriálním senzorem. Vpravo – scéna pořízená senzorem PrimeSense.

Ze získaných množin (automatickou metodou) jsme natrénovali neuronové sítě, které nyní použijeme pro nacházení korespondencí a budeme sledovat jakého výkonu aplikace dosahuje. Zejména nás zajímá, jak si aplikace povede v ideálním scénáři, kdy použijeme senzor, na který je síť naučená. V kontrastu s tím se pak podíváme, jak si aplikace povede, když použijeme jiný senzor než na jaký se síť naučená.

Pro potřeby tohoto testu jsem vytvořil ještě jeden testovací scénář, ve kterém se nachází 54 různých scén a 28 objektů z datasetu ITODD. Experiment je zaznamenán v následující tabulce:

Tabulka 5: Výkon aplikace v závislosti na použité trénovací množině podle použitého senzoru.

Dataset pro testování	Použitá trénovací množina	Recall	Průměrný čas na scénu [s]
tless	Industrial	**1.6%**	20.3
	PrimeSense	**9.6%**	41.5
	Mix	5.9%	26.4
itodd	Industrial	**14.5%**	12.1
	PrimeSense	**6.2%**	60.7
	Mix	11.2%	8.7

Z výsledků vyplívá, že použitím specializované trénovací množiny pro daný senzor lze dosáhnout lepších výsledků rozpoznávání, než kdybychom použili množinu slouženou z dat z více druhů senzorů.

5.3 Architektura sítě

Další parametr, který jsem podrobil testování, byla architektura použité neuronové sítě. Zkoušel jsem sítě s různým počtem vrstev a neuronů, a přitom jsem sledoval schopnost NS naučit se trénovací množinu a počet iterací potřebných k naučení. Jako trénovací množinu jsem použil „Manual B“ s 19 882 prvky viz předchozí kapitola.

Tabulka 6: Výkon aplikace v závislosti na použité architektuře neuronové sítě.

Architektura	Training accuracy	Počet iterací	Recall	Průměrný čas na scénu [s]
66-33-3	81.6%	1636	**3.84%**	6.5
66-33-16-3	83.0%	1614	**5.93%**	8.3
66-66-32-3	87.4%	2504	4.69%	11.8
66-33-32-16-3	85.9%	2360	**6.09%**	10.8

Srovnatelně dobře si zde vedli sítě s architekturou 66-33-16-3 a 66-33-32-16-3. Zajímavé také je, že síť 66-66-32-3, která se byla schopna správně naučit nejvíce prvků, nakonec podala horší výkon než dvě předchozí sítě. To by mohlo znamenat, že je síť přeučená, nebo že moje trénovací množina není dostatečně kvalitní. Síť 66-33-3 je pravděpodobně pro tuto úlohu příliš malá.

5.4 Neuronová síť vs Euklidovská vzdálenost

V tomto testu zkusíme porovnat způsob získávání korespondencí. První způsob je založený na neuronové síti a druhý způsob bude klasické porovnávání dvou klíčových bodů pomocí euklidovské vzdálenosti. Tento druhý způsob má výhodu, že může být akcelerován pomocí k-d stromu, a proto očekávám, že bude oproti neuronové síti řádově rychlejší. V testu budeme sledovat počet nalezených korespondencí, dobu potřebnou k jejich získání a samozřejmě i celkový výkon aplikace. Parametry se liší podle způsobu výběru korespondencí. U neuronové sítě se podíváme na dvě různé trénovací množiny *Real* a *Manual B* viz kapitola 5.1. U euklidovské vzdálenosti budeme volit maximální počet nejbližších sousedů *max_neights* a prahovou hodnotu *sqr_dist*. Bude-li vzdálenost mezi dvěma klíčovými body menší než tato hodnota, budeme je považovat za korespondentní.

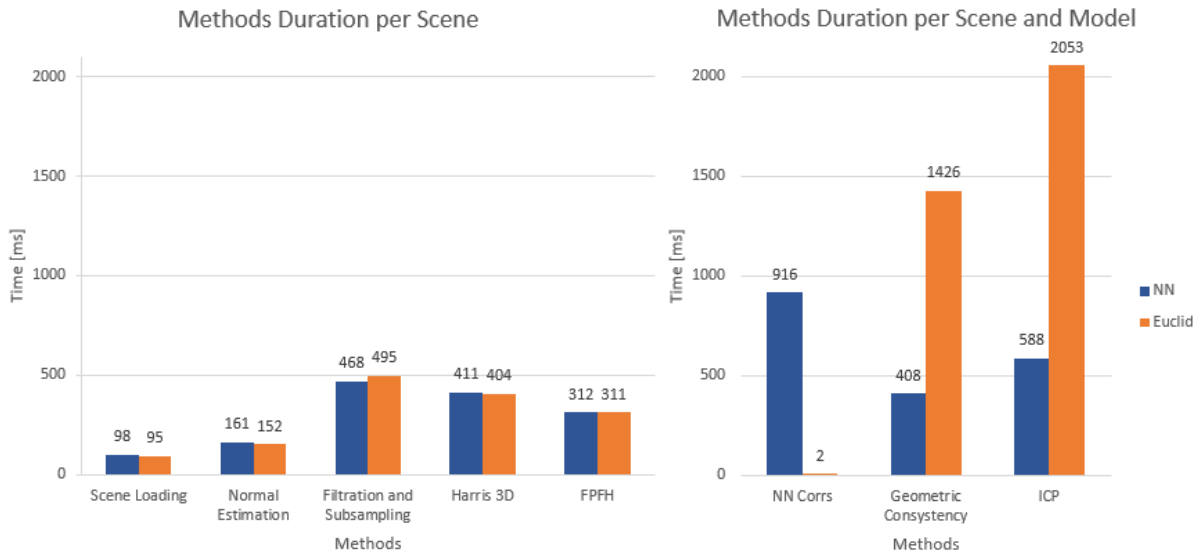
Tabulka 7: Vliv způsobu výběru korespondencí na jejich počet a celkový výkon aplikace.

Způsob výběru korespondencí	Parametry	Průměrný počet korespondencí na		Průměrná doba získávání korespondencí na model a scénu [ms]	Recall	Průměrný čas na scénu [s]
		model a scénu	klíčový bod			
Neuronová síť	Trénovací množina Real	4175	68.5	1050	**6.91%**	31.3
Neuronová síť	Trénovací množina Manual B	1761	28.9	871	**5.63%**	8.2
Euklidovská vzdálenost	sqr_dist = 400 max_neights = 100	74	1.2	2.0	**0%**	1.7
Euklidovská vzdálenost	sqr_dist = 900 max_neights = 130	891	14.6	2.1	**0.21%**	3.7
Euklidovská vzdálenost	sqr_dist = 1200 max_neights = 170	2008	32.9	2.2	2.49%	8.9
Euklidovská vzdálenost	sqr_dist = 1500 max_neights = 200	3474	57.0	2.2	4.09%	21.4

Hodnoty *sqr_dist* a *max_neights* jsem volil podle vlastního uvážení, abych dosáhl rozumného počtu korespondencí na model a scénu a mohl tak provést srovnání s metodou používající neuronovou síť. Z výsledků se zdá, že pomocí neuronové sítě dostáváme o něco kvalitnější korespondence než v případě použití euklidovské vzdálenosti, protože i když je počet korespondencí u euklidovské vzdálenosti stejný nebo dokonce vyšší, tak celkový výkon rozpoznávání nedosahuje výsledků neuronové sítě.

5.5 Doba běhu jednotlivých metod

Nyní se podíváme na dobu trvání jednotlivých metod v rozpoznávacím řetězci, přičemž v prvním případě pracujeme s neuronovou sítí a v druhém případě s euklidovskou vzdáleností, kde $sqr_{dist} = 1500$. Obrázek 28 vlevo ukazuje průměrné trvání metod, které se zpracovávají pouze u scény. A vpravo ukazuje průměrné trvání metod, které jsou zpracovávány pro každý model.



Obrázek 28: Graf znázorňujíc průměrná doba běhu jednotlivých metod na scénu a model.

Z grafů lze jasně vyčíst, že pomocí euklidovské vzdálenosti jsme schopni najít korespondence mnohonásobně rychleji než pomocí neuronové sítě (stejné výsledky viz Tabulka 7). Nicméně neuronová síť vrací korespondence o poznání kvalitnější, to se dá usoudit z toho že následující metody Geometrická konzistence a ICP běže mnohem kratší dobu, a přesto je rozpoznávací výkon aplikace vyšší.

Bližší pohled na trvání rozpoznávání s neuronovou sítí: Kdyby zpracovávání modelů ve scéně (30 modelů v našem případě) probíhalo sekvenčně, celková doba rozpoznávání by byla ~58.8s. Díky paralelnímu zpracování jsme schopni dosáhnout doby rozpoznávání ~8.2s. Na 8 jádrovém procesoru tak dokážeme rozpoznávání v tomto případě urychlit zhruba ~7.2 krát.

Závěr

Cílem této práce bylo vytvořit aplikaci schopnou rozpoznávat 3D objekty a jejich pózu ve scéně a následně modifikovat některý z kroků rozpoznávacího řetězce s cílem jeho vylepšení. Bylo potřeba se podrobně seznámit s problematikou rozpoznávání v mračnu bodů a pochopit jednotlivé kroky nutné k vytvoření funkčního rozpoznávacího systému.

Pomocí metod z knihovny PCL se mi podařilo zprovoznit základní funkční kostru programu schopnou rozpoznávat některé jednoduché objekty ve scéně (viz kapitola 2). Krok, který jsem se následně pokusil vylepšit je hledání korespondujících klíčových bodů mezi objektem a scénou. Jedná se o jeden z klíčových kroků rozpoznávacího řetězce. Pro zjišťování korespondence mezi body jsem rozhodl použít neuronovou síť pomocí knihovny Dlib (viz kapitola 3). Konzolovou aplikaci jsem implementoval v jazyce C++.

V kapitole Experimenty (viz kapitola 5) jsem se zaměřil na hledání některých ideálních parametrů, zejména pro použitou neuronovou síť. Srovnal jsem výkon rozpoznávání aplikace s neuronovou sítí oproti klasickým metodám. A také jsem se zaměřil na různé druhy použité trénovací množiny.

Dle výsledků experimentů, můžu prohlásit, že použití neuronové sítě pro hledání korespondencí má značný potenciál. Zejména s kvalitní trénovací množinou přizpůsobenou konkrétnímu senzoru. Oproti klasické metodě využívající euklidovskou vzdálenost, jsme byli schopni s neuronovou sítí dosáhnout celkově lepších výsledků rozpoznávání. Použití neuronové sítě na úrovni klíčových bodů z ní také činí univerzálně použitelný nástroj pro hledání jakéhokoliv objektu, od něhož máme k dispozici 3D model. Není tedy potřeba neuronovou síť trénovat pro každý model zvlášť. Vše nicméně závisí na kvalitě trénovací množiny. Jak vytvořit co nejkvalitnější trénovací množinu, by tak mohlo být subjektem dalšího zkoumání.

Obecně, s odhlédnutím od hledání korespondencí, by se v aplikaci dalo udělat spoustu dalších vylepšení. Velkým problémem je zde například rychlost rozpoznávání, aplikace není ani z daleka optimalizovaná. Řešením by tak byla paralelizace co nejvíce úloh pomocí GPU. Problematické je také hledání samotných klíčových bodů, některé objekty jich totiž mají příliš mnoho a zpomalují tak běh aplikace a některé objekty naopak nemají vůbec žádné klíčové body a není je možno ve scéně vůbec vyhledat. Další možné vylepšení by mohlo být i použití barevné složky obrazu.

Nakonec můžu říct, že mě práce celkem i bavila, například když jsem přišel s novým nápadem, které by mohl fungovat a zejména potom když opravdu fungoval. Osobně si myslím, že jsem cíle diplomové práce splnil, a dokonce jsem do problematiky vnesl i určitý přínos nových poznatků.

Citovaná literatura

1. **Rusu, R. B. a Cousins, S.** *3D is here: Point Cloud Library (PCL).* Shanghai, China : IEEE International Conference on Robotics and Automation (ICRA), 2011.

2. **King, Davis E.** *Dlib-ml: A Machine Learning Toolkit.* In : Journal of Machine Learning Research, 2009.

3. **Vondrák, Ivo.** *Umělá inteligence a neuronové sítě.* Ostrava : Vysoká škola báňská - Technická univerzita, 1998. ISBN: 80-7078-259-5.

4. **Bentley, Jon Louis.** *Multidimensional binary search trees used for associative searching.* In : Communications of the ACM, 1975. doi: 10.1145/361002.361007.

5. *BOP: Benchmark for 6D Object Pose Estimation.* **T. Hodaň, F. Michel, E. Brachmann, W. Kehl, A. G. Buch, D. Kraft, B. Drost, J. Vidal, S. Ihrke, X. Zabulis, C. Sahin, F. Manhardt, F. Tombari, T.-K. Kim, J. Matas, C. Rother.** München : European Conference on Computer Vision (ECCV), 2018.

6. **Bertram Drost, Markus Ulrich, Paul Bergmann, Philipp Hartinger, Carsten Steger.** *Introducing mvtec itodd-a dataset for 3d object recognition in industry.* In : Proceedings of the IEEE International Conference on Computer Vision Workshops, 2017.

7. **Intel.** Depth Camera D435. *intelrealsense.* [Online] [Citace: 8. Duben 2020.] https://www.intelrealsense.com/depth-camera-d435/.

8. —. Get Started with Intel RealSense Depth Camera. *intelrealsense.* [Online] [Citace: 8. Duben 2020.] https://www.intelrealsense.com/get-started-depth-camera/.

9. **Sipiran, Ivan a Bustos, Benjamin.** *Harris 3D: a robust extension of the Harris operator for interest point detection on 3D meshes.* Springer-Verlag : Vis Comput, 2011. DOI 10.1007/s00371-011-0610-y.

10. **Sojka, Eduard.** *Digitální zpracování a analýza obrazů.* Ostrava : VŠB-TU, 2000. ISBN 80-7078-746-5.

11. **Han, Xian-Feng, a další.** *A comprehensive review of 3D point cloud descriptors.* 2018.

12. **A. E. Johnson, M. Hebert.** *Surface matching for object recognition in complex 3-d scenes.* In : Image and Vision Computing 16 (9-10), 1998. 635–651.

13. **A. Frome, D. Huber, R. Kolluri, T. Bülow, J. Malik.** *Recognizing Objects in Range Data Using Regional Point Descriptors.* Berlin, Springer : European Conference on Computer Vision, 2004.

14. **R. B. Rusu, N. Blodow, Z. C. Marton, M. Beetz.** *Aligning point cloud views using persistent feature histograms.* In : Ieee/rsj International Conference on Intelligent Robots and Systems, 2008. pp. 3384–3391.

15. **R. B. Rusu, N. Blodow, M. Beetz.** *Fast point feature histograms (fpfh) for 3d registration.* In : IEEE International Conference on Robotics and Automation, 2009. pp. 1848–1853.

16. **B. S. Radu, B. Rusu, K. Konolige, W. Burgard.** *Narf: 3d range image features for object recognition.*

17. **F. Tombari, S. Salti, L. D. Stefano.** *Unique signatures of histograms for local surface description.* In : European Conference on Computer Vision, 2010. pp. 356–369.

18. **Rusu, R. B., Blodow, N. a Beetz, M.** *Fast point feature histograms (FPFH) for 3D registration.* Technische Universitat München : IEEE International Conference on Robotics and Automation, 2009.

19. **Chen, Hui a Bhanu, Bir.** *3D free-form object recognition in range images using local surface patches.* Cambridge : International Conference on Pattern Recognition, 2004. ISBN: 0-7695-2128-2.

20. **Horn, Berthold.** *Closed-Form Solution of Absolute Orientation Using Unit Quaternions.* In : Journal of the Optical Society A, 1987. 10.1364/JOSAA.4.000629.

21. **Will Schroeder, Ken Martin, Bill Lorense.** *The Visualization Toolkit An Object-Oriented Approach To 3D Graphics.* Edition 4.1 July 2018.

Elektronické přílohy

Prilohy\Recognizer-manual.pdf – instalační a ovládací manuál k aplikace Recognizer.

Prilohy\Recognizer – obsahuje zdrojové kódy aplikace, vybrané datasety a data neuronové sítě pro otestování funkčnosti. (tato složka a její podsložka build obsahují tyto zdrojové kódy: beta, BopFormat, MachineLearning, Model, Scene, NeuralNetwork, Tools a Viewer)

Prilohy\ModelPreprocesor – obsahuje zdrojový kód a MeshLab skripty pro předzpracování 3D modelů.